SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Darjan Baričević

RAZVOJ KORISNIČKOG SUČELJA U KNOCKOUT.JS

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Darjan Baričević

Matični broj: 44025/15–R

Studij: Informacijski sustavi

RAZVOJ KORISNIČKOG SUČELJA U KNOCKOUT.JS

ZAVRŠNI RAD

Mentor :

Prof. dr. sc. Dragutin Kermek

Varaždin, rujan 2018.

Darjan Baričević

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Rad obuhvaća karakteristike razvoja korisničkih sučelja s naglaskom na dinamičke, odnosno aktivne dijelove sučelja te implementaciju istih u JavaScript biblioteci KnockoutJS. Osim osobina i načina rada same biblioteke, prikazane su osobine korisničkih sučelja te osnovne karakteristike skriptnog programskog jezika na kojem počiva ova biblioteka – javascripta. Isto tako, navedene su sličnosti i razlike, te prednosti i nedostatci ove biblioteke u odnosu na druge okvire i biblioteke koje se koriste za razvoj korisničkih sučelja (Angular, React, Vue). Na kraju rada je izrađena je ogledna aplikacija za kupnju karata za glazbeno-kulturne festivale. Korisničko sučelje aplikacije će sadržavati sve bitne elemente koje će prethodno biti obrađeni u ovome radu, a sam program će koristiti i Bootstrap – programski okvir za dizajniranje web stranica i web aplikacija. O istima će također biti par riječi kroz cjeline.

Ključne riječi: web razvoj; dinamičke osobine; javascript; knockout.js; web aplikacije

Sadržaj

1. Uvod

U posljednjih 20 godina, arhitektura Web-a kakvog danas poznajemo mnogo se promjenila. Za vrijeme trenda „Web 1.0“ još davne 1996. godine i nadalje, bilo je samo oko sto tisuća web stranica na oko 50 milijuna korisnika, što nas dovodi do omjera od 500 korisnika naprema jednoj web stranici. Ovaj trend naziva se još i „read-only web“, odnosno web dostupan samo za čitanje jer su tada u principu ljudi koristili web stranice samo za traženje određenih informacija. Nije postojala potreba za razvoj složenih korisničih sučelja koja bi komunicirala sa korisnikom. Zapravo je postojala samo mala interakcija između web stranice i krajnjeg korisnika, ali nije ni bilo potrebe za njom jer je svrha web stranice bila izvor informacija. [1]

Danas, 20 godina kasnije, broj aktivnih web stranica narastao je na gotovo jednu milijardu, dok je broj korisnika na Web-u oko 2,5 milijarde. Trend se naziva „Web 2.0“ ili „read-write web“ jer podrazumijeva da korisnici generiraju sadržaj i u interakciji su sa samom stranicom. Bitan faktor koji je doprinuo tome je razvoj korisničkih sučelja koja su omogućila takvu interakciju i danas su vrlo popularna, poput responzivnog web dizajna. Zapravo većina stranica koje posjećujemo svakodnevno poput Facebooka, Twittera, LinkedIna, Gmail-a imaju implementirane dinamičke, odnosno responzivne aplikacije i zato gotovo nikada nećemo vidjeti da se stranica osvježava jer se cijeli sadržaj učita odjednom i nema potrebe za otvaranjem više stranica ili „kartica“ na pregledniku. [2] Dok su se davno prije za upravljanje sadržajem web stranice koristile JavaScript naredbe sa manipulaciju DOM-om (engl. *document object model*) i to je u pravilu bilo vrlo teško, dok se danas koriste JavaScript biblioteke i razni okviri (engl. *framework*). Tako je npr. Facebook razvio React biblioteku za izgradnju korisničkih sučelja, Google je razvio Angular okvir za izgradnju web aplikacija, a postoje i brojne druge biblioteke poput Vue.js, jQuery, Redux i Knockout.js. U ovom radu najviše će se govoriti o razvoju korisničkog sučelja u Knockout.js biblioteci, MVVM uzorku koji koristi, praćenju ovisnosti varijabli (engl. *dependency tracking*), te ostalim bitnim stvarima koje će pomoći da se lakše shvati princip rada ove biblioteke.

2. Metode i tehnike rada

S obzirom da je Knockout javascript biblioteka, potreban je uređivač teksta (engl. *text editor*) ili IDE (engl. *integrated development environment*) u kojem će se pisati javascript kôd. U ovom radu korišten je Visual Studio Code iz razloga što je to besplatan uređivač teksta optimiziran za razvoj web aplikacija, ima odličnu podršku za pisanje javascripta, te je izuzetno brz i pregledan.

Biblioteka Knockout.js preuzeta je sa službene stranice ove biblioteke, a instalacija je objašnjena u poglavlju 6.2. Instalacija.

Najbitniji korišten alat za razvoj poslužiteljske strane ogledne aplikacije je XAMPP - distribucija koja u sebi sadrži instalaciju Apache servera, MySQL baze podataka i PHP. Uz pomoć ovog alata moguće je podesiti lokalni server, koristiti lokalnu bazu podataka i tako testirati aplikaciju prvo lokalno umjesto da se sve prvo šalje na server i upisuje direktno u bazu podataka, te izvršava na serveru (više o ovome u potpoglavlju 7.2. Poslužiteljska strana)

U razvoju korisničke strane aplikacije korišten je CSS programski okvir Bootstrap koji će dizajn sučelja uvelike uljepšati i omogućiti aplikaciji da bude responzivna.

3. Web aplikacije

Kako je već navedeno u uvodu, većina web stranica koje korisnici Web-a svakodnevno posjećuju su zapravo rezultat web aplikacija koje su generirale tu stranicu. Drugim riječima, web aplikacija je programski sustav koji generira web stranice i dokumente, a napisan je u nekom od programskih jezika koji se izvršavaju na poslužitelju.[3]

S obzirom da web stranica sama po sebi nema nekakva napredna svojstva sigurnosti kao npr. autorizaciju ili autentikaciju, web aplikacija može imati ulogu zaštite stranice i nadograditi ju sa tim svojstvima. Isto tako, služi za preuzimanje podataka od korisnika i trajno pohranjivanje tih podataka (u bazu podataka ili datoteku), te za prikaz tih podataka drugim korisnicima. Moderne web aplikacije ukljucuju funkcije:

1. web korisničke pošte (engl. webmail)
2. online prodaje, online aukcije (eBay)
3. wiki stranice
4. servise za brzo slanje poruka (engl. *instant messaging services*) poput Whatsapp-a, Viber-a i Facebook Messanger-a.
5. ostale funkcije

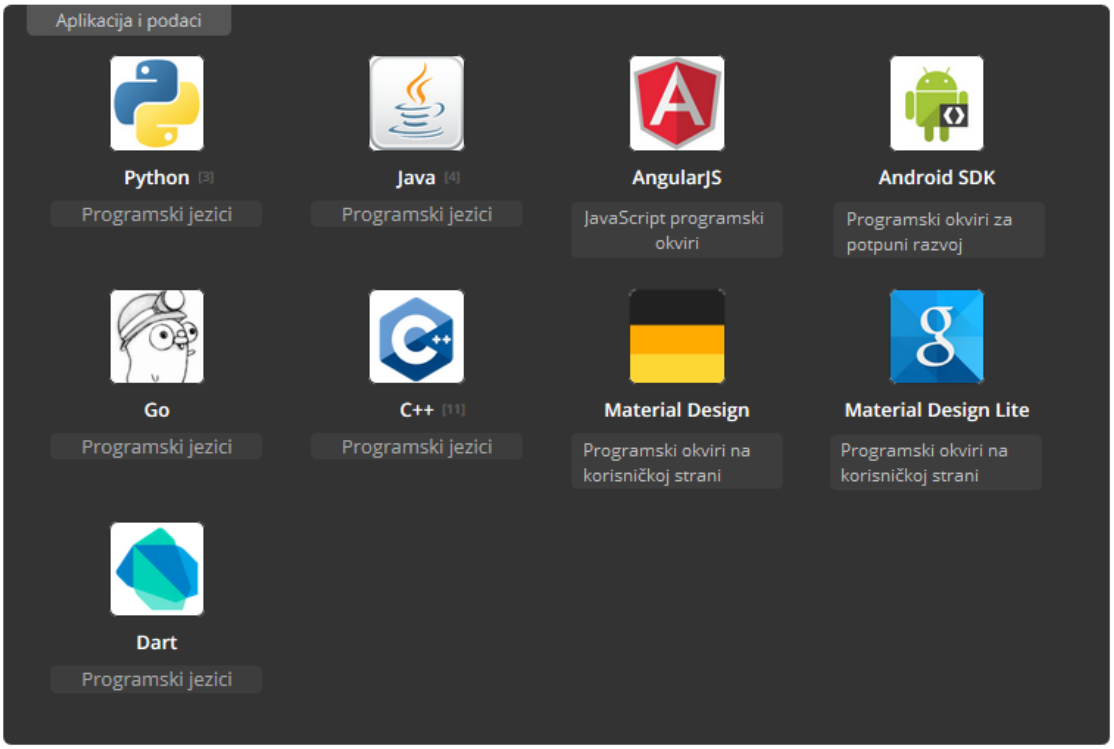
Danas većina korisnika Web-a ne razlikuje pojmove web stranice i web aplikacije i postoji pogrešno shvaćanje da se web aplikacije poistovjećuju sa stolnim aplikacijama, a da je web stranica bilo koja stranica koju korisnik posjeti u pregledniku. To dakako i može biti točno, ali ne mora biti. Web stranica zapravo pruža statički sadržaj svim posjetiteljima, dok web aplikacija pruža dinamički sadržaj ovisno o interakciji sa korisnikom putem korisničkog sučelja. Upravo iz ovog razloga postoji stalna potreba za poboljšanjem korisničke interakcije te razvoja korisničkog sučelja, te se svakodnevno razvijaju novi programski okviri, alati i tehnologije (više u poglavlju 2.2. Korisnička strana aplikacije). Skup svih tih programskih okvira, alata i ostalih tehnologija koje se koriste u razvoju jedne web aplikacije naziva se stog aplikacije ili stog tehnologije (engl. *technology stack*). Oni se stalno mijenjaju i gotovo ih je nemoguće pratiti, ali je moguće u bilo kojem trenutku provjeriti koje tehnologije i alate koriste najpoznatije i najkorištenije aplikacije kao što su Docker, Slack, Dropbox i drugi. Na stranici StackShare (https://stackshare.io – Software and technology used by top companies) moguće je vidjeti cjelokupni stog tehnologije podijeljen u četiri skupine:

1) Aplikacija i podaci (engl. *Application and data*) – odnosi se na tehnologije koje se koriste isključivo za razvoj aplikacije (njezine korisničke i poslužiteljske strane) i pohranu podataka (npr. Javascript i Javascript okviri kao što su Angular i React, PHP, Bootstrap, nginx, Node.js, HTML5, Python i sl.)
2) Uslužni alati (engl. *Utilities*) – alati koji se koriste za posebnu namjenu kao što je web analiza, navigacija, dijeljenje sadržaja i slično (npr. Google Drive, Google Analytics, Postman,

Dropbox itd.)

3) DevOps (engl. *software development and operations*) - odnosi se na alate za integraciju i verzioniranje sadržaja poput GitHuba, Dockera. Ovdje se mogu naći i uređivači teksta kao što je Sublime, WebStorm i Visual Studio Code i drugi.

4) Poslovni alati (engl. *Business tools*) – kao što i samo ime govori, alati za upravljanje projektom i poboljšanje toka rada (npr. Slack, Trello, G Suite i drugi)



Slika 1: *Stog web tehnologije* (vlastita izrada)

Kada se govori u kontekstu programskog inženjerstva, postoji jedan princip, dizajn ili uzorak koji se naziva „odvajanje prezentacije od logike“ (engl. *Separation of concerns*). Princip odvajanja prezentacije od logike je zapravo vrsta „podijeli pa vladaj“ strategije koja dijeli računalni program u odvojene cjeline, od koje je svaka cjelina jedna briga.[4] U ovom slučaju, briga (engl. *concern*) odnosi se na skup informacija koji utječu na kôd računalnog programa. Kako bi se ostvario prethodno navedeni princip, u web razvoju i programskom inženjerstvu općenito razlikujemo dvije strane: korisničku stranu i poslužiteljsku stranu. Podjelom tehnologija i alata na te dvije skupine omogućava odvajanje prezentacijskog sloja (engl. *front-end)* i sloja za pristup podacima (engl. *back-end*). U klijent-poslužitelj arhitekturi (engl *client-server architecture*), klijent se obično smatra korisničkom stranom, a poslužitelj se smatra poslužiteljskom stranom (detaljnije o ovome u poglavljima 2.2 Korisnička strana aplikacije i 2.3. Posližiteljska strana aplikacije). Kao što je vidljivo u slici 1., Facebook koristi React JS programski okvir i PHP na poslužiteljskoj strani, dok Spotify koristi Python i Javu na poslužiteljskoj strani i Bootstrap na korisničkoj. U ovom radu ostati će najveći fokus na alatima i tehnologijama korišteni na korisničkoj strani, odnosno alatima i tehnologijama za razvoj korisničkog sučelja web aplikacije.

3.1. Korisničko sučelje

Čovjek u svojoj svakodnevnici dolazi u interakciju sa raznim kućanskim aparatima, računalima, vozilima, bijelom tehnikom i ostalim stvarima koje on sam pokreće. Svi ti aparati posjeduju korisničko sučelje koji omogućuju tu interakciju i ono je realizirano na jedan ili drugi način. Tako je, naprimjer, u autu korisničko sučelje sačinjeno od: raznih papučica (kvačilo, kočnica, ubrzanje), ručne kočnice, mjenjača brzina, volana i raznih indikatora, te instrumentalnog panela, i stakla koji omogučava pregled okoline u kojem se kreće vozilo. S druge strane, korisničko sučelje jedne perilice za posuđe nije toliko složeno s obzirom da se sastoji samo od gumbova.

Korisničko sučelje (engl. *user interface*) je mjesto susreta odnosno dodira između operatera (osobe) i nekog stroja, sustava ili naprave.[5] U računarstvu se koristi termin grafičko korisničko sučelje (engl. *graphical user interface*) ili skraćeno GUI, dok se kod strojeva poput automobila koristi termin strojno sučelje (engl. *machine interface*). Grafičko korisničko sučelje realizira interakciju s korisnikom koristeći razne grafičke elemente, tekstualne poruke ili obavijesti. Zapravo sve web aplikacije imaju grafičko korisničko sučelje i spadaju u GUI aplikacije, dok se isto ne mora odnositi i na stolne aplikacije. Primjer je Git alat za integraciju i verzioniranje sadržaja koji se može koristiti u obliku naredbenog retka (Git Bash) ili Git GUI alata koji osim naredbenog retka imaju i korisničko sučelje (SourceTree, GitKraken, SmartGit i drugi) radi lakše vizualizacije toka rada (više o ovome u poglavlju 2.1.2. Podjela korisničkih sučelja). Nadalje, elementi grafičkog korisničkog sučelja su:

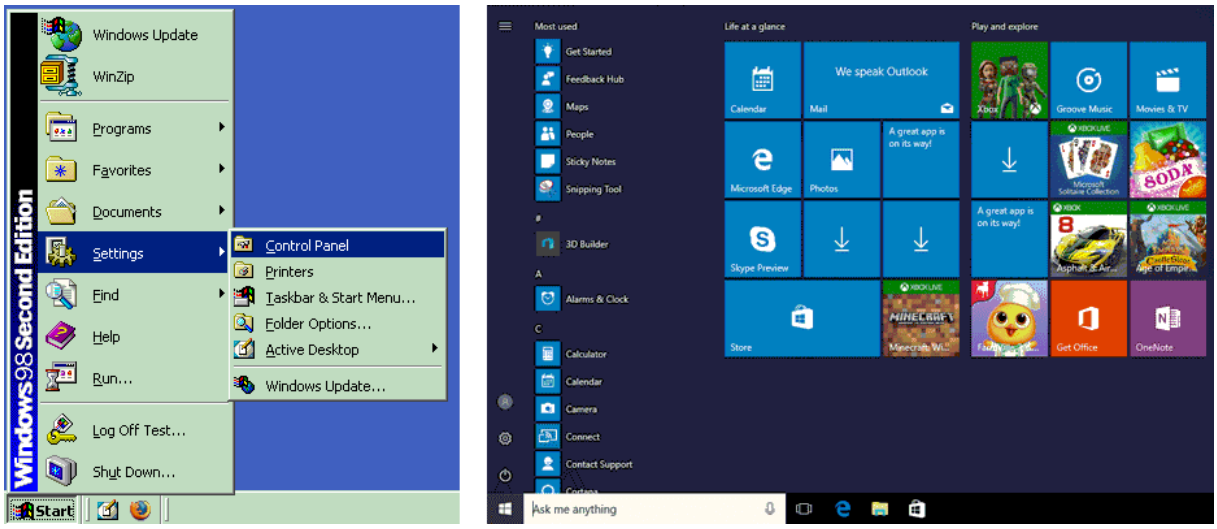
1) Prozor (engl. *window*)
2) Ikone (engl. *icons*)
3) Izbornik (engl. *menu*)
4) Pokazivač ili kursor (engl. *cursor*)

To su zapravo osnovni vizualni elementi koji se nalaze u WIMP paradigmi (engl. *window-icon-menu-pointer*). Danas se koriste još brojni drugi elementi u post-WIMP sučeljima i njih se ponajviše veže uz razvoj Android ili iOS korisničkog sučelja. Biblioteka ili kolekcija biblioteka koje sadrže grafičke kontrole elemente (engl. *widgets*) za konstriuranje grafičkog korisničkog sučelja naziva se widget toolkit.

3.1.1. Povijest

Godine 1960. američki inžinjer Douglas Engelbart radio je na projektu 'Augmentation of Human Intellect', odnosno povećanju ljudskog znanja u interakciji sa informacijskim tehnologijama. U tu svrhu razvijeno je računalo 'oN-Line System' ili skraćeno NLS. Takvo računalo prvo je posjedovalo kursor i nekoliko prozora na sučelju koji su se bazirali na hipertekstu. Engelbart je

bio toliko fasciniran tim računalom da je odlučio to primjeniti u svojoj kompaniji - Xerox PARC. To je dovelo do razvoja osobnog računala pod nazivom 'Alto' koje je imalo bitmap zaslon i to je zapravo prvo računalo sa grafičkim korisničkim sučeljem. Zatim je kompanija 1981. godine razvila radnu stanicu pod nazivom Xerox Star koja nije doživila pretjerani uspjeh, ali je utjecala na budući razvoju sučelja kod Apple-a na svojim Macintosh računalima. Kasnije je i samu ideju preuzeo Microsoft u prvim verzijama Windows računala. Od njih se kasnije razvio i KDE (engl. *K Desktop Environment*) i GNOME (engl. *GNU Network Object Model Environment*) - grafička korisnička sučelja koja se koriste u UNIX operacijskim sustavima, pretežito Linux-u.



Slika 2: *Usporedba Windows 98 i Windows 10 korisničkog sučelja* (vlastita izrada)

S lijeve strane (Slika 2.) nalazi se korisničko sučelje Windows 98 operacijskog sustava koje je zapravo bilo 2. izdanje Windowsa i u sebi nije sadržavalo neke elemente bez kojih danas ne možemo zamisliti korisničko sučelje, a to su: sat, kalendar, prozor za upravljanje glasnoćom, tipkovnica za zaslonu (engl. *on-screen keyboard*). Unatoč tim svim nedostatcima, sučelje je i dalje bilo veliki napredak u odnosu na Windows 95 korisničko sučelje koje nije imalo ni preglednik, pozadinske teme, traku za adresu i još brojne druge mogućnosti. Vidljivo je da ono tada nije imalo puno boja niti je izgledalo previše elegantno i oku ugodno u usporedbi sa današnjim Windows 10 sučeljem (Slika 2.).

Danas se sve više i više teži elegantnijem, intuitivnijem i inovativnijem korisničkom sučelju koji će korisniku olakšati rad ne samo na računalu, već i na tabletu i mobilnom uređaju. Neusporedivo je lakše korištenje računala danas nego 1970-ih godina, odnosno u doba UNIX operacijskih sustava koji su bili poprilično otežani za uporabu s obzirom da nisu imali elemente korisničkog sučelja, već samo naredbeni procesor (engl. *shell*) i datotečni sustav. U okviru koncepata modernih korisničkih sučelja, fokus je također na sadržavanju mnogobrojnih elemenata, opcija i mogućnosti uz istodobnu preglednost zaslona. Tako današnji zasloni na sebi mogu sadržavati elemente poput: klizača aktivnog ekrana (engl. *screen slider*), padajućeg izbornika sa dostupnim bežičnim mrežama (engl. *wi-fi dropdown menu*), virtualne pozadine (engl. *virtual desktops*) i sl. Još jedna zanimljiva mogućnost koja se pojavila u Windows 10 sustavu je

upravljanje glasom (engl. speech recognition) koje omogućava da upravljamo računalo glasom, odnosno bez korištenja miša ili tipkovnice i znatno olakšava interakciju sa sučeljem.

3.1.2. Podjela korisničkih sučelja

Naravno, nisu sva korisnička sučelja jednaka. Kategoriziraju se prema načinu na koji korisnik djeluje na njih. Tako Heathcote (2004.)[6] dijeli korisnička sučelja u 3 glavne grupe:

a) Sučelje sa komandnom linijom (engl. *CLI, command-line interface*)

b) Grafičko korisničko sučelje (engl. *GIU, graphical-user interface*)

c) Prirodno korisničko sučelje (engl. *NUI, natural user interface*)

Ovdje se mogu naći i korisnička sučelja bazirana na formama (engl. *form-based interface*) i sučelja upravljana izbornikom (engl. *menu-driven interface*, ali ona u teoriji spadaju u grafička korisnička sučelja. Sučelje sa komandnom linijom koristi FreeDOS i interakcija s njim moguća je samo pomoću pisanja raznih komandi. To danas nije praktično za uporabu i više se koriste grafička korisnička sučelja. Prirodna korisnička sučelja pak zahtijevaju interakciju dodirom, pokretima ili govorom. Smatraju se najjednostavnijim za korištenje i upravo iz tog razloga se nazivaju prirodnim sučeljima. U razvoju mobilnih tehnologija nastoji se omogućiti i olakšati interakciju pokretnima i govorom, pa se tako npr. na Android sustavu nedavno pojavio Google Gestures aplikacija koja omogućava korištenje mobilnog uređaja sa „švrljanjem“ po zaslonu. Karakteristike svakog sučelja ukratko su navedene u tablici 1.

Sučelje sa komandnom linijom (CLI)	**Grafičko korisničko sučelje (GUI)**	**Prirodno korisničko sučelje (NUI)**
Određeno kodom	Prilagođeno za samoučenje	Direktno
Strogo definirano	Metaforičko	Intuitivno

Tablica 1: Karakteristike tipova korisničkih sučelja [6]

3.1.3. Dizajn korisničkog sučelja

U prošlom odlomku spomenuto je da se danas više teži razvoju intuitivnijih grafičkih korisničkih sučelja, što se zapravo jednim dijelom postiže dobrim dizajnom, a drugim dijelom dobrim odabirom alata i tehnologija korištenih u razvoju sučelja. Kaže se da se zapravo dizajn realizira dobrim tehnologijama, stoga u svakom slučaju ne ide jedno bez drugog.

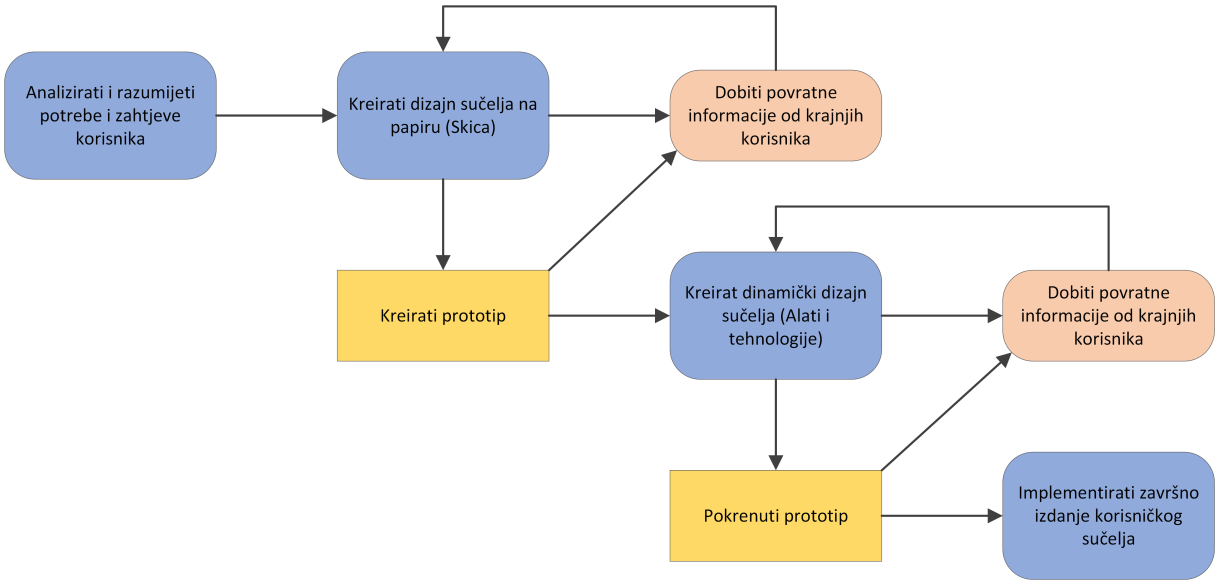
Dizajn korisničkog sučelja (engl. *user interface design*) odnosi se na dizajniranje korisničkih sučelja svih strojeva i aparata, te programa (engl. software) s naglaskom na što bolje maksimiziranje korisničkog iskustva te iskoristivosti sučelja. Valja napomenuti da na povećanje iskoristivosti bitno utječu elementi grafičkog dizajn i tipografije. Poznato je i da je dobar dizajn

korisničkog sučelja jedan bitan i neizostavan dio u mnogim projektima i njihovim rješenjima, počevši od računalnih sustava i CNC strojeva, pa sve do automobila i zrakoplova. Stoga, grafički dizajneri se specijaliziraju u 3 područja:

1) Programski ili softverski dizajn (engl. *software design*)
2) Web dizajn (engl. *web design*)
3) Industrijski dizajn (engl. *industrial design*)

Prvo što čovjeka asocira uz dizajn najčešće je: arhitektura, skica, nekakav oblik, konstrukcija tijela i sl. Međutim, programski dizajn kao i industrijski dizajn su puno više od toga. Ono najčešće uključuje rješavanje problema i planiranje kroz programsko rješenje, dok industrijski dizajn podrazumijeva primjenu dizajna na proizvode u masovnoj proizvodnji. To se postiže tako da se odvoji dizajn proizvoda, u smislu njegovih osobina, dimenzija i oblika, od fizičkog nastajanja tog proizvoda tj. njegove proizvodnje. Neovisno o kojem području govorili, korisničko sučelje se uvijek dizajnira na jedan od sljedećih načina:

a) Skiciranjem – nema određenih metoda i tehnika, jednostavno prostoručno crtanje skice željenog korisničkog sučelja
b) Primjenom digitalnog dizajna – korištenje alata kao što su npr. Adobe Creative Cloud alati (Adobe Fireworks, Adobe XD i sl.)
c) Prototipiranjem – stvaranje vjerodostojnog prikaza korisničkog sučelja kreiranjem nekoliko okvirnih prikaza (engl. mockup) ili maketa (engl. wireframe). Danas postoje brojni alati poput Axure RP ili proto.io koji omogućuju da prototip bude interaktivan kao i prava web stranica ili web aplikacija, bez ikakvog pisanja programskog koda.



Slika 3: *Proces dizajna korisničkog sučelja* [7]

Sukladno prethodno navedenim metodama i tehnikama, svaki razvoj korisničkog sučelja počinje sa analizom korisničkih potreba i zahtjeva. Nakon toga je uobičajeno napraviti nekakav grubi izgled sučelja, najčešće u obliku skice na papiru. To je zapravo statički dizajn sučelja nakon kreiranja koji služi da krajnji korisnici imaju dojam kakva je uopće ideja sučelja. Nakon povratnih informacija korisnika, kreira se prototip i nakon toga mu se dodaju dinamička obilježja. Tek nakon što krajnji korisnici evaluiraju sučelje u tom obliku, može se započeti sa izrada završnog izdanja korisničkog sučelja.

3.1.4. Web dizajn

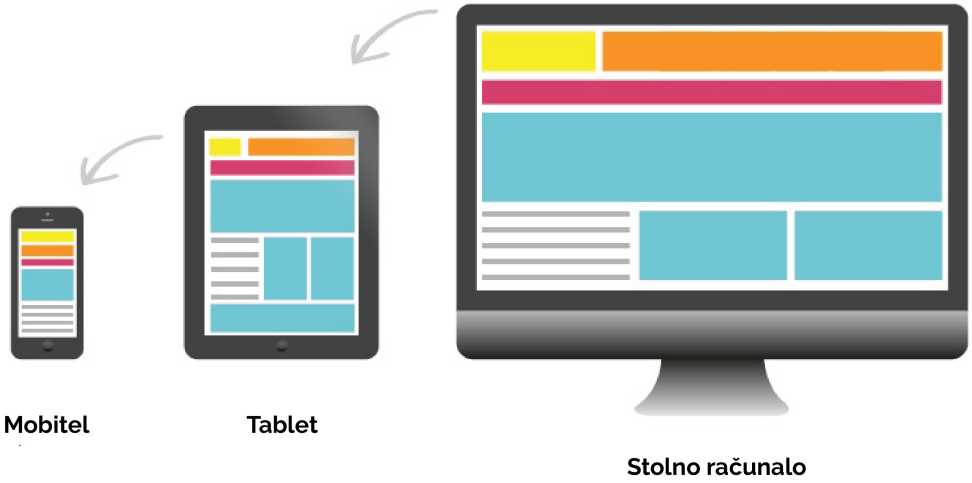
Postoje brojne definicije o tome što je zapravo web dizajn i svaka od njih je točna na neki način. Razlog tomu je taj što na web dizajn utječu brojna druga područja poput programiranja, mrežama računala, knjižničarstva, grafičkog dizajna i razvoj tehnologije općenito. Prema Powell (2002.g)[8], postoji nekoliko gledišta u okviru kojih možemo definirati web dizajn:

1) Sadržaj (engl. *content*) – odnosi se formu i organizaciju sadržaja web stranice, te način prezentacije sadržaja
2) Vizualni elementi (engl. *visuals*) – odnosi se na sve ono što je vidljivo na web stranici, a najčešće je rezultat zajedničkog djelovanja HTML-a, CSS-a i Javascripta. Također može uključivati i Flash reprodukciju sadržaja, ali to je u današnje vrijeme jako zastarjelo i praktički se ne koristi.
3) Tehnologija (engl. *technology*) – ovo se može odnositi na cjelokupni tehnološki stog, na jedan programski okvir, programski jezik, ili bilo kakav alat koji utječe na interaktivnost stranice i korišten je u razvoju na poslužiteljskoj ili korisničkoj strani.
4) Prikaz (engl. *delivery*) – odnosi se na performanse stranice i brzinu kojom se sadržaj prenosi kroz protokol
5) Svrha ili namjena (engl. *purpose*) – odnosi se na namjenu web stranice kao takve u smislu nekog ekonomskog problema ili rješenja. Ovo gledište uvijek se treba uzeti u obzir neovisno na koji način definirali web dizajn.

Jedan način na koji bi se mogle sve komponente prikazati kao jedna cjelina, je u obliku piramide. Zamislimo piramidu iz bočnog pogleda sa korisnicima na jednoj strani i dizajnerima na drugoj strani, te svrsi stranice na vrhu piramide. Sve tri točke, odnosno sva tri vrha povezuje sadržaj kojeg prikazuju dizajneri krajnjim korisnicima putem formi i drugim vizualnim elementima. Razlog postojanja te interakcije je zapravo namjena te stranice, koja je kao što je već navedeno pretežito nekakav ekonomski problem ili njegovo rješenje.

Većina ljudi definira web dizajn sa gledišta sadržaja i vizualnih elemenata web stranice te smatraju da je najveći izazov web dizajna isključivo usklađenost sa gomilom današnjih web pre-

glednika koji podržavaju ili ne podržavaju neke web standarde.[9] To je samo jedan od izazova, a u današnje vrijeme još uvijek postoje web preglednici koji nisu u potpunosti podržali sve elemente HTML verzije 4, isto kao što gotovo svaki preglednik podržava različite opcije i animacije korištene u CSS verziji 3. Drugi izazov koji se veže uz pojam web dizajna je razvijanje web stranice koja se prilagođava svakom zaslonu. To se naziva odzivni ili prilagodljivi web dizajn (engl. responsive web design). Autor svake web stranice ili aplikacije svjestan je da će ta ista stranica ili aplikacija biti prikazivana na nekoliko desetaka ili pak stotina zaslona različitih dimenzija. Kako on može osigurati da će se na svakom uređaju, odnosno zaslonu njegova web stranica prikazati onako kako i treba biti prikazana? To jest, da će obrasci i polja za unose biti prilagođene širine, da će uvijek svi elementi potrebni elementi biti prikazani, da će slike biti maksimalne širine ekrana uređaja i ono najbitnije – da će brzina učitavanja sadržaja uz sve navedene zahtjeve i dalje biti optimizirana. Kreiranje jedinstvene prilagodljive stranice umjesto nekoliko stranica za svaki uređaj je svakako najbolje rješenje kada uzmemo vrijeme i trošak razvoja u obzir (Bryant i Jones, 2012.g).



Slika 4: *Prilagodba sučelja na različitim dimenzijama* [10]

Odzivni ili prilagodljivi web dizajn je pristup u web dizajnu koji teži tomu da se web stranica prikazuje jednako dobro na svakom uređaju ili zaslonu. Taj pristup najviše podrazumijeva sljedeće:

1) Korištenje relativnih mjernih jedinica kao što su postotci (%) ekrana umjesto apsolutnih veličina kao što su pixeli (px) ili centimetri (cm)
2) Prilagodljive slike čije su dimenzije isto iskazane u postotcima kako bi se spriječilo prelijevanje slike, odnosno prikaz slike izvan elementa u koji je postavljena
3) Korištenje „CSS Media queries“ opcije unutar CSS-a koja zapravo učitava različite stilske upute ovisno o dimenziji ekrana

Prije 10-ak godine možda i nije bilo tolike potrebe za ovim pristupom jer je većina prometa na

internetu odvijala na stolnim računima. No, danas se više od 50% prometa na internetu odvija preko mobitela i potreba za prilagodljivim web dizajnom sve je veća i veća.

3.2. Korisnička strana aplikacije

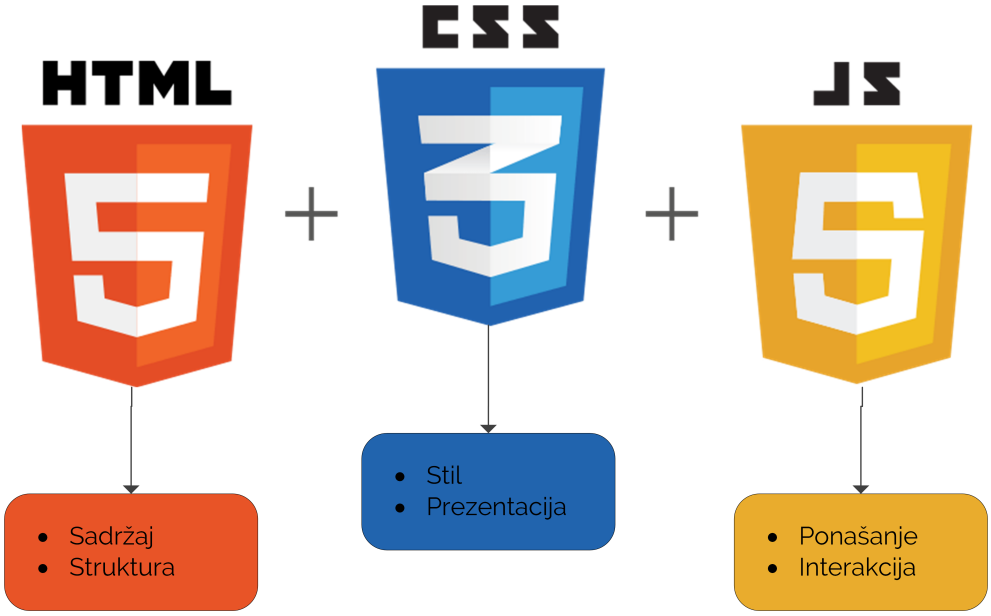
Kao što je već navedeno u poglavlju 2.1, razvoj aplikacije je radi odvajanja prezentacije i logike podijeljen u dva dijela: razvoj na korisničkoj strani i razvoj na poslužiteljskoj strani. Razvoj korisničke strane aplikacije (engl. front-end development) zapravo omogućava interakciju sa aplikacijom ili web stranicom, a veže se uz korištenje 3 temeljne tehnologije:

1) HTML (engl. *HyperText Markup language*) – jednostavni jezik oznaka koji se koristi za kreiranje hipertekstualnih dokumenata koji su neovisni o platform, a sastoji se od elemenata koji mogu biti prepoznati unutar HTML grafičkih i znakovnih preglednika i kao takvi biti prikazani na specifičan način.[11] Razvio ga je Tim-Berners Lee 1991. godine, a danas je ovaj jezik temelj svakog web razvojnog procesa. Kroz povijest postojalo je nekoliko verzija jezika, a posljednja verzija je HTML5.

2) Kaskadne stilske upute (engl. *CSS, Cascading Style Sheets*) - stilski jezik koji se koristi za opisivanje HTML elemenata i smatra se jednom od ključnih funkcionalnosti razvoja na korisničkoj strani. Ovaj jezik zapravo određuje kako će stranica izgledati u smislu boja, veličine fonta, veličine slika, raspored elemenata i ostalo. Inicijalnu verziju jezika predložio je Håkon Wium Lie 1994. godine, a prihvaćena je 1996. godine od strane World Wide Web konzorcijuma. Posljednje verzija je CSS3, a ujedno je i najznačajnija zbog novih opcija prikaza: grid i flex. Te su dvije opcije danas najmoćniji koncepti CSS-a jer omogućuju beskonačno mogućnosti rasporeda elemenata u jednodimenzionalnom ili dvodimenzionalnom prikazu.

3) Javascript ili skraćeno JS - skripti programski jezik koji se izvršava u web pregledniku na strani korisnika. Razvila ga je kompanija Netscape 1995. godine, a posljednja verzija je JS5 (više o ovome u poglavljima 3. i 4.)

Kako su ove tri tehnologije korištene zajedno najlakše se može dočarati sa analogijom jednostavne rečenice. Rečenica se, kao što je uvelike poznato, sastoji od: imenica, pridjeva i glagola. U slučaju razvoja na korisničkoj strani, imenice se odnose na HTML i njene osnovne elemente, a pridjevi na kaskadne stilske upute jer daju vizualna svojstva imenicama, odnosno HTML-u. Analogno tome, JavaScript dodaje dinamiku cijeloj web stranici. Osim ovih, postoje i drugi brojni alati i tehnologije koje se koriste u razvoju na korisničkoj strani. Dakako, odabir tih alata i razumijevanje ideje iza odabira koji od njih je najbolje koristiti u određenim slučajevima čini razliku između generičke stranice i pravilno dizajnirane te skalabilne web stranice.[12] Kako bi razvili brže i skalabilne stranice, oznake i stil i logika moraju biti usklađene. Usklađenost se najviše postiže odvajanjem prezentacije od logike, a tim pristupom znatno se olakšava posao na poslužiteljskoj strani. Uz ovo sve, mora se uzeti u obzir i informacijska struktura koja se neprestano mijenja te zahtjeva napredna znanja i uporabu novih tehnologija.



Slika 5: *Temeljne tehnologije u razvoju na korisničkoj strani* (vlastita izrada)

Osim web razvojnih programera, uz web razvoj vežu se i još dva specijalizirana područja: dizajner korisničkog sučelja (engl. *UI designer, user interface designer*) i dizajner korisničkog iskustva (engl. *UX designer, user experience designer*). Tako dizajner korisničkog iskustva najviše utječe na ono kakav osjećaj daje proizvod, odnosno sučelje. Zapravo rješava probleme (svrhu ili namjenu proizvoda) na način da učini proizvod što intuitivnijim, a to zahtjeva povratne informacije korisnika te može potrajati i nekoliko stotina iteracija dok se ne utvrdi najoptimalnije korisničko iskustvo. S druge strane, dizajner korisničkog sučelja zadužen je za fizički dojam korisničkog sučelja i prikaz istog. Izrađuje prototipe, razne animacije i prilagođava sučelje raznim zaslonima u teorijskom smislu. Valja napomenuti da u oba područja gotovo nema programiranja, već je fokus na radu u raznim UI/UX alatima, analizi korisnika i razvoju sadržaja (engl. *content development*).

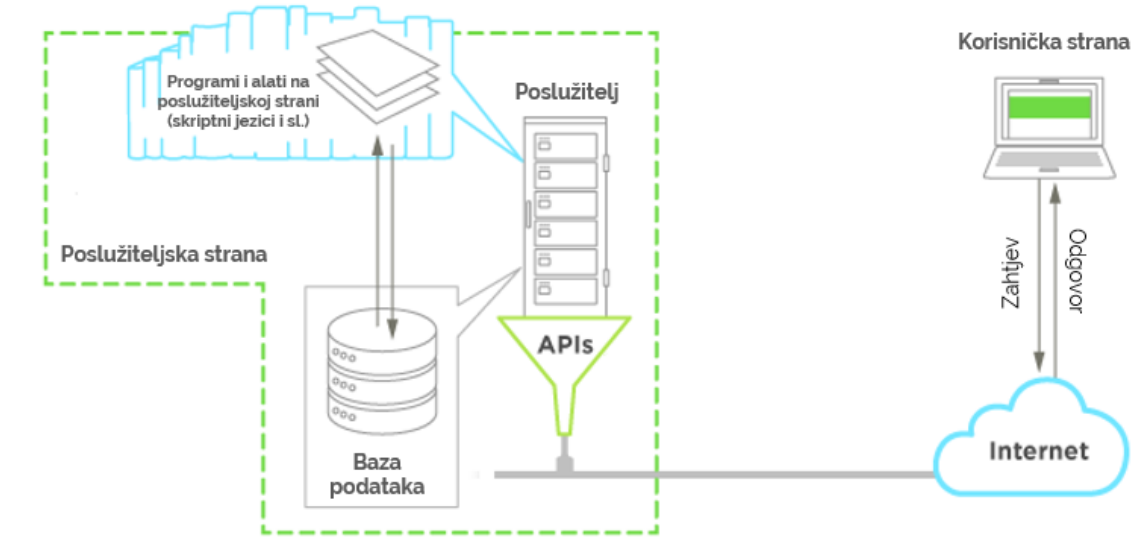
3.3. Poslužiteljska strana aplikacije

Već je rečeno da je korisnička strana zapravo sve ono što se prikazuje korisnicima u web pregledniku. Ovdje se postavlja pitanje: gdje je zapravo druga strana aplikacije, gdje je fizički smještena ta aplikacija?

Svaka web aplikacija ili web stranica općenito smještena (i instalirana) na udaljenom poslužitelju (engl. *server*). Poslužitelj je ni više ni manje nego stvarno računalo koje u pravilu mora biti snažnijih specifikacija kako bi moglo posluživati što više klijenata. Pa tako jedan klijent može koristiti više poslužitelja, a jedan poslužitelj može posluživati više klijenata. Takva se arhitektura naziva klijent-poslužitelj (engl. *client-server architecture* arhitektura, ili danas još zahtjev-odgovor (engl. *request-response*) arhitektura jer klijenti šalju zahtjeve prema poslužitelju koji im šalje odgovore i tako implementira svoje usluge. Tip poslužitelja na kojem su smještene web

aplikacije naziva se web poslužitelj, a postoje još i poslužitelji baze podataka, datotečni poslužitelji, e-mail poslužitelji i dr. Danas postoje poslužitelji kapaciteta nekoliko stotina terabajta sa procesorima od nekoliko desetaka jezgri koji mogu posluživati tisuće klijenata, a isto tako postoje sustavi koji zahtjevaju tisuće ovakvih poslužitelja u svojim podatkovnim centrima.

Razvoj na poslužiteljskoj strani (engl. *back-end web developmenent*) podrazumijeva rad na arhitekturi i logici web stranice i svega što se događa na poslužitelju općenito. Ako je web stranica hrana koja vam treba biti poslužena, onda je programer na poslužiteljskoj strani kuhar koji će ju pripremiti, a programer na korisničkoj strani konobar koji će ju dostaviti.



Slika 6: *Arhitektura poslužiteljske strane* [13]

Sukladno prethodno navedenoj analogiji, korisnička strana ne može stalno raspolagati podacima koji se nalaze na poslužiteljskoj strani. Korisnik putem sučelja zapravo komunicira sa drugom stranom i može slati ili tražiti podatke od poslužitelja. U mogućnosti je povremeno tražiti te podatke korištenjem AJAX (engl. *asynchronous JavaScript And XML*) poziva ili korištenjem aplikacijskih programskih sučelja (engl. *API, application programming interface*) (više o ovome u poglavljima 4. i 5.). Nakon toga, komunikacija između poslužitelja i baze podataka ostvaruje se korištenjem skriptnih programskih jezika.

Najčešće korišteni skriptni jezici na poslužiteljskoj strani (engl. *server-side scripting languages*) su: PHP, Ruby, Perl Python i dr. U nastavku slijede neke od karakteristika skriptnih programskih jezika:

a) kod se izvršava na poslužitelju i ugrađen (engl. *embedded*) je u kod stranice

b) napravljen je da može komunicirati sa bazom podataka

c) pokreće se na poziv, npr. kad se dogodi AJAX poziv tada se program izvršava i vraća podatke

d) mogu nadograditi web stranicu sa naprednim opcijama sigurnosti poput autorizacije, autentikacije, aktivacije korisnika, blokiranja računa i sl.

e) temelj su razvoja aplikacijskih programskih sučelja koji omogućuju komunikaciju sa drugim aplikacijama

Što se baza podataka tiče, u uporabi su najviše relacijske baze podataka poput MySQL-a, Oracle, PostgreSQL i Microsoft SQL Server. Naravno, kako bi se aplikacija uopće mogla pokrenuti, potrebno ju je izvršiti na web poslužitelju. Ovdje se govori o web poslužitelju u smislu programa, a ne sklopovlja. Najpoznatiji besplatni web poslužitelj je Apache HTTP Server, a dostupan je u XAMPP distribuciji zajedno sa PHP5, phpMyAdmin platformom te MariaDB bazom podataka. Ovakve distribucije omogućuju korisnicima da vrlo brzo postave poslužitelj i započnu razvoj web tehnologija.

U usporedbi sa web razvojem na korisničkoj strani, poslužiteljska strana je po mnogim mišljenjima dosta kompleksnija. Korisnička strana možda zahtjeva viziju i smisao za dizajn sučelja, ali poslužiteljska strana zahtjeva šira znanja: kako funkcionira HTTP, kako radi web poslužitelj i web servisi, rad s većim količinama podataka i bazama podataka, poznavanje raznih programskih okvira na poslužiteljskoj strani i sl.

4. Programski jezik JavaScript

4.1. Povijest

Javascript (JavaScript) ili skraćeno JS je skriptni programski jezik koji je 1995. godine razvio Brendan Eich, razvojni programer koji je tada radio u Netscape Communications kompaniji. Početna ideja je bila da se jezik nazove Mocha, iako je prva verzija izdana pod imenom LiveScript. Razvoj jezika je bio strelovit unatoč brojnim kritikama tadašnjih razvojnih programera zbog nedostatka planiranja i vizije. Netscape je imao jedan od tada najboljih web preglednika Navigator, i u jednoj verziji iste godine LiveScript je konačno dobio ime Javascript. Riječ "Java" unutar Javascripta nije slučajna, ali ta dva jezika osim sintakse zapravo nemaju puno sličnosti. Kako bi se u startu uklonile ovakve nedoumice, navedene su temeljne razlike:

- JavaScript se koristi u razvoju na korisničkoj strani, dok se Java koristi u razvoju na poslužiteljskoj strani
- JavaScript se izvodi dinamički, dok se Java izvodi statički
- Java je objektno-orijentiran jezik i zasnovan je na klasama, dok je JavaScript zasnovan na prototipovima

Netscape je 1996. godine objavio da su Javascript prijavili u Ecma - privatnu, internacionalnu i neprofitnu organizaciju za standardne u području informacijske i telekomunikacijske tehnologije.[14] Tako je Ecma objavila prvo izdanje ECMA-26 specifikacije, koja je ustvari bila Javascript. Iz tog razloga se danas ovaj standard (jezik) zove ECMAscript ili skraćeno ES nakon čega slijedi sufiks verzije jezika. Posljednje veće promjene u jeziku donijela je verzija ES5, a posljednja radna verzija je ES6 koja je izdana 2015. godine pod imenom Harmony i danas je standard i temeljna tehnologija u web razvoju.

Danas je JavaScript korišten pod licencom od strane Mozilla Foundation neprofitne organizacije. Stoga je Mozilla Foundation razvio zajednicu Mozilla Developers Network, ili skraćeno MDN (MDN Web Docs). Na njoj se mogu pronaći najkvalitetnije, najnovije i najpouzdanije informacije u okviru web razvoja. Isto tako, na dnevnoj bazi se objavljuju i ažuriraju novi članci, pojašnjenja (engl. *tutorials*), vodiči i reference kako bi članovi bili u toku sa najnovijim alatima i tehnologijama. Početnicima na području web razvoja preporuča se uvijek prvo tražiti informacije i upute na ovoj zajednici kako bi bili sigurni da će uvijek dobiti točne informacije i da će moći dobiti reference na prave izvore.

4.2. Svojstva jezika

Glavno obilježje ovog jezika je to da je danas postao standard u području web razvoja i prema tome ima univerzalnu podršku od strane većine web preglednika. Također, bitne karakteristike jezika su sljedeće:

a) Deklarativni i strukturirani jezik - podržava gotovo sve elemente unutar C sintakse kao što su if instrukcije, while petlje, do while petlje i ostalo. Izuzetak su područja rada programa (engl. *scope*) koje se moglo odrediti isključivo sa ključnom riječi var sve do izdanja ES5 kada su se pojavile let i const naredbe za definiranje područja rada unutar bloka naredbi i unutar funkcija. Isto tako valja napomenuti i da Javascript razlikuje instrukcije (engl. *statements*) i izraze (engl. *expressions*).

b) Slabo povezivanje podataka - već je rečeno da je JavaScript dinamički jezik. To znači da, kao i većina skriptih jezika, ima slabo povezivanje podataka što znači da varijable nemaju deklaraciju tipa podataka. Ovo dozvoljava varijablama da mjenjaju tip podataka po potrebi tijekom izvršavanja programa. Ponekad ovo svojstvo olakšava pisanje programa jer interpreter ne javlja greške kod pridruživanja vrijednosti pojedinoj varijabli, a ponekad otežava jer se u tom pretvaranju tipa podataka može kriti logička greška.

c) Zasnovan na objektima (prototipovima) - većina elemenata u JavaScript-u su objekti, ali to ne znači da je jezik objektno-orijentiran. Ne koristi klase, nasljeđivanje, sučelje i ostala principe objektno-orijentiranih jezika nego za većinu tih svojstava ima svoje prototipe kojima se ostvaruje to svojstvo.

d) Funkcijski jezik - to znači da se funkcije tretiraju kao objekti. U prijevodu, moguće je da jedna funkcija unutar sebe poziva drugu funkciju, proslijeđuje funkciju kao parametar, pridružuje funkciju kao vrijednost neke varijable i drugo. Isto tako, funkcije mogu unutar sebe imati svojstva kao što je .ajax ili .bind u jQuery okviru koje su ustvari i same funkcije. Ako se unutar jedne funkcije definirana druga funkcija, to se onda naziva učahurena funkcija.

Od ostalih karakteristika valja napomenuti da JavaScript podržava dozvoljene izraze (engl. *RegExp, regular expressions*). Dozvoljeni izraz je uzorak (niz znakova) koji opisuje ili zamjenjuje drugi niz znakova (engl. *string*) u skladu sa nekim pravilima. Npr, ako želimo neko mjesto u stringu zamijeniti sa bilo kojim pozitivnim brojem, koristiti ćemo izraz {n} koji zamijenjuje bilo koji pozitivni broj. Tako postoje i izrazi koji zamijenjuju slova (velika i mala), niz slova, niz znakova, početak ili kraj niza pa čak i cijele riječi. Ovo je vrlo moćan koncept koji omogućava manipulaciju teksta i provjeru svih mogućih unosa. Ukoliko se želi provjeriti da li je uneseni e-mail u pravilnom formatu, koristiti će se isključivo dozvoljeni izrazi.

4.3. Kako pisati JavaScript?

U prethodnom poglavlju rečeno je da se JavaScript izvršava u pregledniku, odnosno na korisničkoj strani. To znači da se JavaScript programi u web pregledniku proslijeđuju zajedno sa cijelim sadržajem stranice (HTML-om i CSS-om).

Tako postoje dva načina da se uključi JavaScript kod u sadržaj stranice, a oba načina zahtjevaju korištenje `<script></script>` oznake:

a) 1. način - Pisanje JavaScript naredbi unutar samog HTML-a:

```
<script>
    ... Ovdje se pišu JavaScript naredbe ...
</script>
```

b) 2. način - Uključivanje .js izvorne datoteke

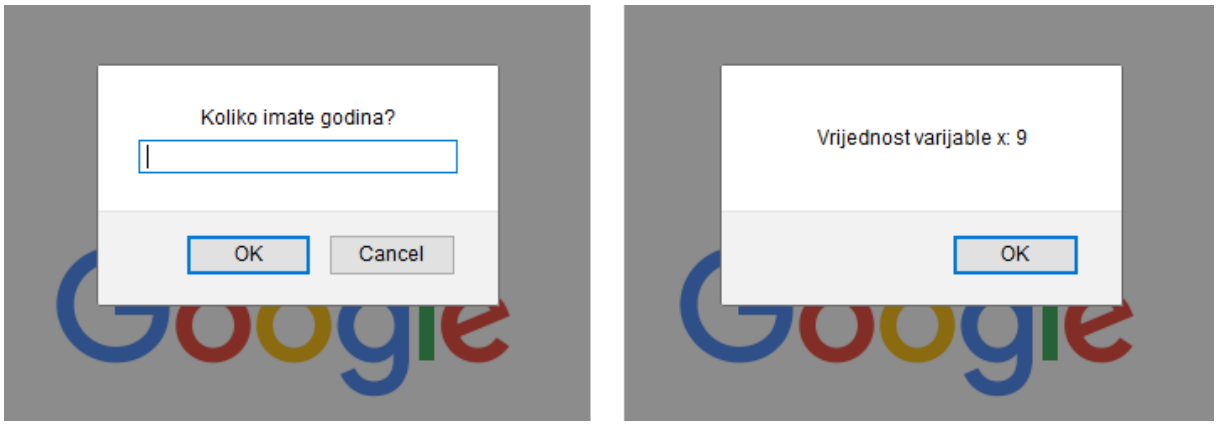
```
<script src="mojKod.js"></script>
```

U oba načina moguće je na bilo kojem mjestu definirati skriptu. Ako se uključuje .js izvorna datoteka u zaglavlju HTML dokumenta, tada će se sav kod izvršiti slijedno baš kao u proceduralnim jezicima. Ako je skripta definirana na nekom drugom mjestu unutar HTML datoteke, web preglednik će prvo učitati sav HTML sadržaj prije te oznake, izvršiti skriptu i nastaviti dalje sa učitavanjem sadržaja. Međutim, uvijek se preporuča korištenje drugog pristupa iz brojnih razloga. Kako navodi Ballard (2015.)[15], prednosti u postavljanju JavaScript koda u odvojenu .js datoteku su sljedeće:

- Kada se ažurira kod u odvojenoj datoteci, taj kod se automatski i ažurira u svakom HTML dokumentu koji je uključio tu skriptu. Ovo je naročito bitno u kontekstu Javascript programskih okvira i biblioteka koje se uključuju u program kao vanjske .js datoteke.
- Kôd unutar HTML dokumenta je čišći i pregledniji te lakši za održavanje
- Web stranica je time nešto optimiziranija u smislu brzine jer web preglednik sprema uključenu vanjsku datoteku u priručnu memoriju (engl. *cache memory*), te ju odmah ima pripremljenu za ponovnu uporabu

JavaScript naredbe se također mogu pisati unutar Web Console dodatka kojeg danas imaju svi moderni web preglednici. To je zapravo interaktivni prevoditelj (engl. *interpreter*), nešto poput Python Shell okruženja. Taj dodatak omogućava pisanje i automatsko prevođenje JavaScript naredbi. Najčešće se koristi za isprobavanje pojedinih naredbi ukoliko razvojni programer nije siguran da li ispravno rade u njegovom programu. Isto tako, može se koristiti za upravljanje objektnim modelom dokumenta na kojem se korisnik trenutno nalazi, ali više o tome u poglavlju 3.4.

S obzirom da se JavaScript koristi na korisničkoj strani, primjereno je da se ulazno-izlazne operacije obavljaju preko sučelja. Pri tome misli se na forme za unos, padajuće izbornike, navigacijske trake, paragrafe, labele i sve ostale HTML elemente. Ako se želi forsirati način sličan C++/C# jeziku (preko konzole), ovdje se to može učiniti koristeći `prompt()` i `alert()` naredbe. Naredba `prompt()` će prikazati prozor sa zeljenim naslovom i poljem za unos (Slika 7, lijevo), dok će naredba `alert()` prikazati prozor sa zeljenim tekstom ili varijablom (Slika 7, desno).



Slika 7: *Prikaz funkcija prompt() i alert()* (vlastita izrada)

Treba svakako napomenuti da ovo nisu jedne od naredbi koje bi trebale prijeći u naviku tijekom programiranja. Koriste se isključivo za svrhe testiranja, kada se želi u bilo kojem trenutku provjeriti stanje neke varijable, kada se želi provjeriti da li je uopće došlo do nekog dijela programa, kada nije sigurno da li polja za unos rade ispravno i sl.

4.4. Sintaksa jezika

Kao i kod svakog drugog proceduralnog i strukturnog jezika temeljenog na C sintaksi, središnji dio se sastoji od:

1) Varijabli, konstanti, doslovni izraza (literala)
2) Izraza i operacija
3) Funkcija
4) Objekata (klasa)

U sljedećim potpogavljima će biti ukratko objašnjen svaki pojedini segment središnjeg dijela kroz primjere i isječke kodova.

4.4.1. Varijable, konstante, literali

Varijable se deklariraju sa ključnom riječi `var` ili `let` koja se pojavila nakon ES6.

```
var x = 6;
x = 6;
let x = 6;
```

U prethodnom primjeru zapravo su sve 3 linije identične jer se u svakoj deklarirala varijabla `x` i pridružila joj se vrijednost 6. Jedina razlika između ključnih riječi `var` i `let` je u području rada. Kada varijablu definiramo sa ključnom rijei `var`, ona je dostupna u cijeloj toj funkciji dok

je varijabla definirana sa ključnom riječi `let` dostupna samo do kraja sljedećeg bloka naredbni, odnosno slijedeće vitičaste zagrade.

Literali su kao što im i ime govori doslovne vrijednosti. Kada se nekoj varijabli želi dodijeliti fiksna vrijednost, onda se ona naziva literal. Varijabli možemo dodijeliti i identifikator druge varijable, a razlika između literala i identifikatora je u tome što je literal fiksna vrijednost, a varijabla promjenjiva.

```
var x = 7
var y = 20.22
var x = var y
```

Primjetite da u prošlom primjeru nije pisana točka sa zarezom (engl. *semicolon*) na kraju svake linije. To je zato što ju Javascript automatski dodaje na kraj reda i ona je opcionalna osim u slučajevima kada se izričito želi naglasiti kraj linije.

```
var x = 9; var y = 3
```

Znakovne nizove (engl. *string*) možemo definirati unutar jednostrukih ili dvostrukih navodnika.

```
var str = "Marko Markić";
var ime = 'Pero Perić';
```

Konstante se u Javascriptu definiraju pomoću ključne riječi const, a njih koristimo kad želimo deklarirati neku konstantnu ili nepromjenjivu varijablu.

```
const pi = 3.14;
const TEMP = 30;
```

Treba svakako napomenuti da je Javascript osjetljiv na velika i mala slova (engl. *case sensitive*). Stoga, ako se umjesto ključne riječi `const` napiše `consT`, preglednik će to pročitati kao varijablu sa nazivom `consT` i izbaciti grešku ukoliko ona nije definirana. Prema Brown (2016.), pravila oko imenovanja varijabli su sljedeća:

- Nazivi varijabli moraju počinjati sa slovom, znakom dolara ($) ili podvućenom crtom (_)
- Nazivi varijabli se mogu sastojati samo od elemenata navedenih u prošloj natuknici (slova, znaka $ i znaka _)
- Unicode znakovi su dozvoljeni (kao npr, π ili λ)
- Nazivi varijabli ne mogu biti rezervirane riječi, odnosno ključne riječi ili nazivi već postojećih i ugrađenih funkcija

Jedan od bitnih koncepata koji još nije spomenut je pisanje komentara. Vrlo su važni za razvoj programa u timovima jer daju razne informacije i smjernice, a u JavaScriptu se pišu na sljedeći način:

```
// Jednolinijski komentar
ili
/*
    Višelinijski komentar
 */
```

4.4.2. Tipovi podataka

U Javascriptu, svaka vrijednost je primitivnog tipa (engl. *primitive*) ili objektnog tipa. Primitivni tipovi podataka spremaju primitivne vrijednosti, odnosno nepromjenjive (engl. *hard-coded*) vrijednosti. Takvi tipovi podataka nemaju nikakve metode i svojstva. Naprimjer, niz znakova `'Informatika'` će uvijek biti `'Informatika'`. Primitivni tipovi podataka su sljedeći:

1) Number (brojevi)
2) String (nizovi znakova)
3) Null
4) Boolean
5) Undefined

Mnoge razvojne programere uvijek muči razlika između Null vrijednosti i Undefined vrijednosti, ne samo u JavaScript jeziku i bez obzira da li su početnici ili nisu. Po zvuku, obje vrijednosti odaju da su nešto nedefinirano, pa koja je razlika onda? Promotrite sljedeći primjer:

```
var Primjer;
alert(Primjer); // Ispisuje undefined

var Primjer = null;
alert(Primjer); // Ispisuje null
```

Undefined znači da je varijabla deklarirana, ali joj još uvijek nije pridružena vrijednost. S druge strane, null je pridruženo stanje koje reprezentira varijablu bez vrijednosti.

Za razliku od primitivnih tipova koji mogu poprimiti samo jedan oblik, objektni tipovi mogu poprimiti više oblika. Njih se može predočiti kao kolekciju imenovanih vrijednosti. Sve vrijednosti zapisane su u obliku **ime : vrijednost**, a odvajaju se sa zarezom. U sljedećem primjeru prikazan je objekt koji predstavlja stvarnu osobu:

```
var Osoba =
{
    Ime: "Pero",
    Prezime: "Perić",
    Dob: 21
}
```

Ključevi "Ime", "Prezime", "Dob" zapravo se nazivaju svojstva (engl. *properties*). Osim svojstva, objekti mogu imati i metode odnosno svojstva koje sadrže definiciju funkcije unutar sebe. Metode su zapravo akcije koje mogu biti izvršene nad objektom unutar kojeg su definirane. Tako je u prošlom primjeru mogla biti definirana metoda "VratiImePrezime" koja bi vraćala spojeno ime i prezime kako ih ne bi morali dohvaćati pojedinačno.

Drugi način na koji se kreiraju objekti je pomoću ključne riječi `new`. Tako bi prethodni objekt iz prošlog primjera kreirali na sljedeći način:

```
var osoba = new Object();
osoba.Ime = "Pero";
osoba.Prezime = "Perić";
osoba.Dob = 21;
```

Ovaj koncept kreiranja novog objekta je vrlo važan jer su objekti, kao što je već navedeno, vrlo promjenjivi. To znači da im se pristupa preko reference, a ne preko vrijednosti. Ako se naprimjer kreira nova varijabla `novaOsoba` i pridruži joj se objekt `osoba` iz prošlog primjera, tada nije kreirana kopija objekta `osoba` nego je kreirana varijabla `novaOsoba` preko koje će se referencirati taj objekt. To je zapravo još uvijek isti objekt, i ako se dogodi promjena nad njime, također će biti vidljiva ako se objekt referencira preko nove varijable.

JavaScript ima i nekoliko ugrađenih objektnih tipova, a to su:

1) Array (Polja)
2) Date (Datum)
3) RegExp
4) Map
5) Set

`Array` objektni tip podataka omogućava kreiranje jednodimenzionalnih ili višedimenzionalnih polja, asocijativnih polja i sl. `Date` objektni tip reprezentira datum u željenom obliku (milisekunde, datum u obliku niza znakova, YYYY/MM/DD HH:i:s oblik i sl.). `RegExp` je bio objašnjem u prošlom potpoglavlju, a `Map` i `Set` služe za pohranjivanje jedinstvenih vrijednosti primitivnog ili objektnog tipa i ne koriste se previše.

4.4.3. Kontrola toka podataka

Kada bi se JavaScript programom htjelo simulirati nekakav stvarni svakodnevni događaj, ponavljajuću akciju ili aktivnost onda bi se svakako moralo koristiti neke od elemenata za kontrolu toka podataka. JavaScript podržava nekolicinu instrukcija za kontrolu toka podataka koje programima dodaju interaktivnost, a to su:

1) Blok instrukcija (engl. *block statement*)
2) `while`, `for` i `do while` petlje
3) Provjera uvjeta (engl. *if statement*)
4) `switch` instrukcija

Blok instrukcija koristi se za grupiranje više instrukcija odjednom, a JavaScript ju tretira kao jednu jedinicu odnosno instrukciju.

```
{
    instrukcija_1;
    instrukcija_2;
    .
    .
    instrukcija_n;
}
```

Što bi se dogodilo kada bi u prethodnom primjeru umjesto ovih točki bili prazni redovi? Apsolutno ništa, jer JavaScript ignorira razmake, tabulatore i prijelaze u novi red. To daje slobodu programerima da koriste koliko god razmaka i tabulatora im je potrebno, i da formatiraju svoj kod po želji sve da bi ga učinili elegantnijim i čitljivijim.

Ponavljanje niza instrukcija realizira se korištenjem petlji. Petlja `for` koristi se kada se zna točno koliko puta se želi ponoviti blok naredbi. Naprimjer, ako se želi ispisati svi brojevi od 1 do 100, to će se učiniti na sljedeći način:

```
for (var i=1; i<=100; i++)
{
    alert(i);
}
```

Međutim, ako se ne zna koliko točno puta će se izvesti blok naredbi, onda će se koristiti `while` ili `do while` petlja. Jedina razlika je u tome što će se u `do while` petlji blok naredbi izvršiti minimalno jedanput, čak i ako je uvjet neistinit. S druge strane, kod `while` petlje prvo se provjerava uvjet pa tek onda se kreće na izvršavanje naredbi.

```
while (dob < 18)
{
    dob = prompt("Koliko imate godina?");
}
```

Prethodni isječak koda je samo primjer korištenja `while` petlje i ne bi se trebao koristiti u praksi jer može uzrokovati usporavanje ili čak zamrzavanje korisničkog sučelja.

Instrukcije `if`, `if..else`, `if..else if..else` izvršavaju neku instrukciju ili blok instrukcija ako je određeni uvjet ispunjen. Kod prve se može definirati samo ono što će se dogoditi ako je uvjet ispunjen, dok `if..else` i `if..else if..else` omogućavaju da se definira i što će se izvršiti ako uvjet nije ispunjen i omogućavaju definiranje drugih uvjeta.

```
if (ocjena === 5)
{
    alert("Odličan");
}
```

Kada bi se ispred prethodne `if` instrukcije definiralo `var ocjena = "5"`, da li bi uvjet bio ispunjen? U ovom slučaju ne bi jer je u uvjetu korišten logički operator '===' koji u JavaScriptu provjerava da li je tip podataka jednak i da li su vrijednosti jednake, dok logički operator '==' provjerava samo dali su vrijednosti u uvjetu jednake i na ovo se uvijek mora paziti.

4.4.4. Izrazi i operatori

Izraz je svaki pravilan skup varijabli, operatora i literala koji u konačnici rezultira nekom vrijednošću. Ovisno o tipu vrijednosti kojom mogu rezultirati, izraze dijelimo na:

a) Aritmetičke
b) String
c) Logičke

Kako bi se uopće moglo doći do nekog izraza, potrebni su operatori koji će proizvesti taj izraz. Isto kao i kod izraza, postoje aritmetički operatori koji su standardni kao i kod svakog drugog programskog jezika: zbrajanje (+), oduzimanje (-), mnozenje (*), cjelobrojno dijeljenje (/), ostatak cjelobrojnog dijeljenja (%) i drugi. U JavaScriptu se nizovi znakova mogu zbrajati (engl. *string concatenation*), a rezultat toga je novi niz znakova:

```
str = "Java" + "Script" // Nova vrijednost 'str' je JavaScript
```

Što se tiče logičkih operatora podržanih od strane JavaScripta, postoje samo 3 i to su: AND (&&), OR (||) i NOT (!). Bilo koji od navedenih operatora može se pisati tekstualno kao što je napisano, ili znakovno kao što je napisano u zagradi. Logički operator AND služi za povezivanje više uvjeta u instrukcijama za kontrolu toka podataka i vraća `true` samo u slučaju ako su svi uvjeti ispunjeni. S druge strane, operator OR vraća `true` je bilo koji od uvjeta ispunjen. Operator NOT se može primjeniti i na varijablu tipa boolean kako bi dobili suprotnu vrijednost ili na bilo koji uvjet kako bi ga negirali.

4.4.5. Funkcije

Bez funkcija gotovo je nezamislivo programirati na korisničkoj ili poslužiteljskoj strani te razvijati web aplikacije uopće. Bilo bi moguće naravno, ali bi svakako bilo otežano jer bi pisanje koda

bilo jako usporeno, a sam kôd neuredan i otežan za održavanje.

Funkcija je skup instrukcija koji se pokreće kao jedna cjelina, odnosno kao potprogram (Brown, 2016.). Njen je zadatak da pretvori ulazne podatke koji se nazivaju parametri ili argumenti, u izlazne podatak koji se naziva rezultat funkcije.

```
function Pozdrav()
{
    console.log("Pozdrav svima");
}
```

Primjetite da kod JavaScripta nije potrebno definirati tip funkcije ovisno o rezultatu koji vraća (integer, string, boolean), isto kao što nije potrebno definirati ulazne parametre i rezultat koji vraća funkcija. U prethodnom primjeru kreirana je funkcija koja će svaki put kad ju se pozove sa `Pozdrav()` ispisati "Pozdrav svima" u Web Console dodatak (spomenuto u Poglavlju 3.3.). Naredba `console.log()` vrlo je korisna za potrebe testiranja jer omogućava da se u Web Console karticu ispiše neka vrijednost bez da iskoči prozor i uspori se rad programa kao u `alert()` naredbi.

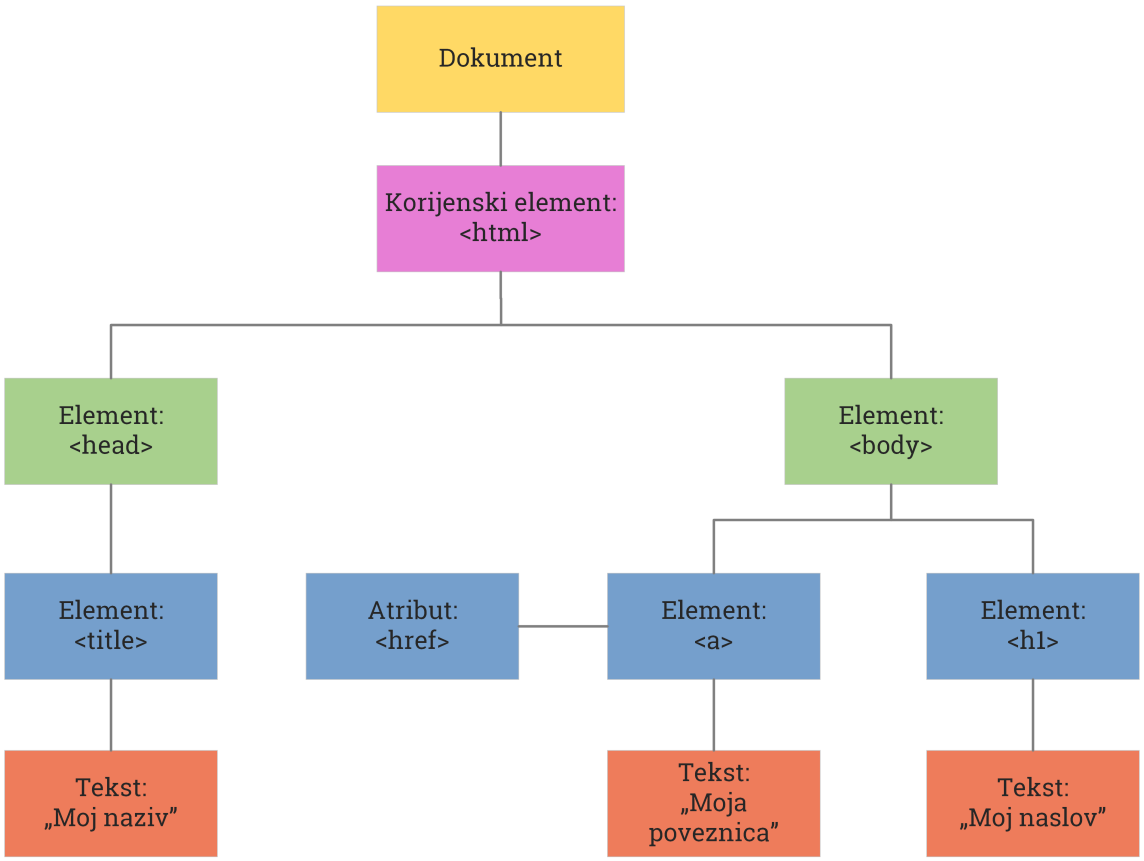
```
function Povrsina (a, b) {
    if (x > 3 && y == true)
    {
        return a * b;
    }
    else
    {
        console.log("Duljine stranica ne smiju biti negativne!");
    }
}
```

U prethodnom primjeru prikazana je funkcija koja za ulazne parametre prima duljine stranica a i b nekog pravokutnika i vraća površinu tog pravokutnika, a u slučaju da su stranice negativne duljine ispisuje poruku greške. Sama suština funkcija je u tome da ih se može iznova koristiti i pozivati bilo gdje u programu, čak i u drugim funkcijama. Takva se funkcija naziva ugnježdena funkcija (engl. *nested function*).

4.5. Objektni model dokumenta

Kada se web stranica učita u web pregledniku, preglednik prvo kreira objektni model dokumenta i u njega pohranjuje sve informacije o tom dokumentu. Taj model zapravo predstavlja stablo sa čvorovima kod kojeg svaki čvor predstavlja jedan objekt.

Kada je kreiran objektni model dokumenta, tada JavaScript dobiva potpunu moć upravljanja tim modelom. Što to znaći? To znaći da je tada moguće mijenjati taj model, odnosno moguće je



Slika 8: *Objektni model dokumenta* (vlastita izrada)

izmijenjivati HTML elemente i njihov sadržaj, maknuti i dodavati neke stilske upute i sve ostalo vezano uz sam dokument. To je moguće jer je objektni model dokumenta zapravo programsko sučelje za pravilno formatirane HTML i XML dokumente. [16]

```
var naslov = document.getElementById("naslov");
naslov.style.color = "pink";
```

U prethodnom je primjeru prvo dohvaćen naslov sa identifikatorom 'naslov' i zatim mu je promjenjena boja u rozu. S obzirom da se naslov mora nalaziti unutar dokumenta, dohvaćen je preko čvora `document` koji ima svojstvo `getElementById`. Na taj se način mogu dobiti i elementi preko njihovog imena klase, imena oznake i sl. Osim `document` tipa podataka, čvorovi mogu biti i tipa: `element`, `nodeList`, `attribute` i `namedNodeMap`.

4.6. Upravljanje događajima

Za registraciju na nekoj web stranici prvo je potrebno popuniti obrazac sa poljima za unos. Takvi obrasci moraju biti interaktivni i korisniku olakšavati cijeli proces registracije. Naprimjer, kada se klikne na polje za unos ono promjeni boju da korisniku naglasi fokus na tom polju, kada se klikne na gumb "Završi" treba se pojaviti tekstualni element sa potvrdom o uspješnoj registraciji. Isto tako, ako je neki unos neispravan onda treba automatski treba korisniku dati

do znanja tako što promjeni okvir, boju pozadine i sl. Sva ova dinamika rezultat je JavaScript događaja, a mnoge od njih moguće je upravljati putem rukovatelja događaja (engl. *event handlers*).

Svakom elementu moguće je pridružiti događaj, odnosno rukovatelj događaja. To se najjednostavnije može učiniti preko `addEventListener` metode koja kao prvi parametar prima naziv događaja, a kao drugi parametar funkciju koja će se izvršiti.

```
var tipkaZavrsi = document.getElementbyId('tipkaZavrsi');
tipkaZavrsi.addEventListener("click", function()
{
    alert("Registracija uspješna!");
});
```

Bilo je svakako moguće prvo odvojeno napisati funkciju koja će se izvršiti i onda ju samo referencirat kao drugi parametar metode, ali je u prethodnom primjeru sama funkcija napisana unutar ove metode. Takva se funkcija zove anonimna funkcija jer nema referencu i postoji samo unutar napisanog područja rada. Najviše se koriste u radu sa događajima jer na jednoj web stranici mogu biti stotine događaja koji rade jednu određenu stvar i nema nikakve svrhe odvojeno pisati i imenovati funkciju za svaki od tih događaja. Iako je definiranje rukovatelja događajem unutar HTML-a moguće i sasvim ispravno, nije preporučeno kao dobra programerska navika jer nije poželjno logiku pisati unutar HTML-a. (Ballard, 2015).

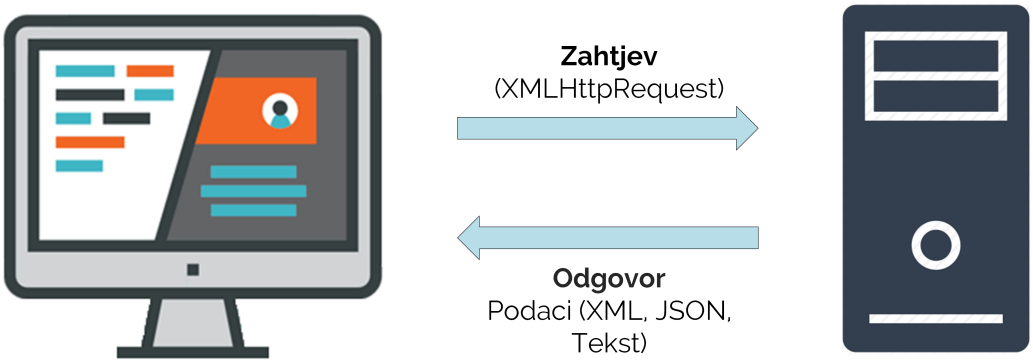
Osim `Click` događaja, objektni model dokumenta raspolaže i sa oko 200 drugih događaja koji se aktiviraju na neku akciju korisnika ili preglednika. Kategoriziraju se prema tipu akcije koja pokreće događaj, a najpoznatiji su: događaji tipkovnice i miša, događaji formi, događaji fokusa, događaji izvora pa sve do CSS događaja. Upravo zbog ovoga je upravljanje događajima gotovo najmoćniji koncept razvoja na korisničkoj strani jer čini stranicu interaktivnijom, olakšava korisničku interakciju i uljepšava sam boravak na web stranici i samim time poboljšava korisničko iskustvo.

4.7. AJAX

U 2. poglavlju objašnjena je korisnička strana aplikacije, odnosno ono što se prikazuje korisnicima u web pregledniku i poslužiteljska strana aplikacije, odnosno korijen i sva logika aplikacije na drugom kraju. Sada se postavlja pitanje: kako korisnička i poslužiteljska strana mogu komunicirati? Recimo da poslužiteljska strana u jednom trenutku mora znati što se događa na korisničkoj strani, odnosno što je korisnik kliknuo kako bi znala što treba vratiti korisničkoj strani i kako bi se to prikazalo bez osvježavanja sučelja.

AJAX je skup tehnologija korištenih na korisničkoj strani za razvoj asinhronih web aplikacija. Omogućava web aplikaciji da šalje i prima podatke od poslužitelja asinhrono, odnosno u poza-

dini i bez osvježavanja sučelja. Neka vas ne zavarava riječ "skup" u prošloj rečenici, AJAX je zapravo jedna tehnologija koja objedinjuje više tehnologija (HTML, CSS, JavaScript), a piše se kao dio JavaScripta.



Slika 9: *AJAX zahtjev i odgovor* (vlastita izrada)

Na korisničkoj strani kreira se zahtjev (XMLHttpRequest) i definira što ova strana traži od poslužitelja, odnosno koje podatke. Tada poslužitelj prima zahtjev, i korisničkoj strani vraća tražene podatke u obliku XML-a, JSON-a ili običnog teksta. Danas se više koristi JSON (engl. *JavaScript Object Notation*) jer je to prirodni oblik za JavaScript i lakše se izvode operacije čitanja datoteke, iteriranja kroz datoteku i sl. Napomenuto je već da se svi ovi zahtjevi šalju asinhrono, odnosno u pozadini tako da aplikacija ne mora čekati na odgovor nakon slanja zahtjeva nego nastavi s radom. Također, moguće je i osvježavanje korisničkog sučelja bez osvježanja same stranice što dodatno poboljšava korisničko iskustvo i ključno je za razvoj modernih jednostranih web aplikacija. Tako se, naprimjer, na Facebook aplikaciji sve novosti osvježavaju konstantno bez da je korisnik osvježio stranicu.

```
var tipkaZavrsi = document.getElementbyId('tipkaZavrsi');
tipkaZavrsi.addEventListener("click", DohvatiTekst);

function DohvatiTekst()
{
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function()
    {
        if (this.readyState == 4 && this.status == 200)
        {
           document.getElementById("stari_tekst").innerHTML = this.responseText;
        }
    };
    xhttp.open("GET", "skripta.php", true);
    xhttp.send();
}
```

U prethodnom je primjeru pomoću DOM-a dohvaćena tipka `tipkaZavrsi` i dodan joj pris-

tuškivač događaja koji će okinuti funkciju `DohvatiTekst` kada se klikne na tipku. Početak svakog AJAX poziva je kreiranje `XMLHttpRequest` objekta (zahtjeva) i pozivanje metode `onreadystatechange` koja je zapravo rukovatelj događaja i poziva se kod svake promjene stanja. U ovom slučaju, kada se promjeni stanje prvo će se provjeriti da li je dokument učitan (`this.readyState == 4`) i da li je odgovor od poslužitelja potvrdan (`this.status == 200`). Ako je, učitat će odgovor poslužitelja u obliku teksta (`this.responseText`) i pridružiti ga odabranom paragrafu. Metoda `open` prima 3 parametra: tip zahtjeva (GET/POST), lokaciju prema kojoj šaljemo zahtjev (URL) i asinhroni način slanja (true ili false). Lokacija prema kojoj šaljemo je najčešće PHP skripta koja se nalazi na poslužitelju u kojoj je definirano kako treba reagirati kada dobije određeni zahtjev.

Uočljivo je da ovoliko naredbi za slanje jednog AJAX zahtjeva nije baš pogodno uz razvoj većih i složenijih aplikacija na kojima gotovo sve što se događa na sučelju mora biti bez osvježavanja same stranice. Takve aplikacije mogu imati i do nekoliko stotina AJAX poziva u svojem JavaScript kodu, što opet dovodi do nepreglednog koda koji je težak za održavanje. To je jedan od razloga zbog kojeg se pojavila potreba za razvojem JavaScript programskih okvira, a više o njima u poglavlju 4.

5. Programski okviri

5.1. Općenito

Većina računalnih korisnika su bar jednom u životu svjesno ili nesvjesno koristili ili instalirali programski okvir u nekom obliku, bilo da je riječ o manjem ili većem okviru. Ovo se ponajviše odnosi na jedan od najvećih programskih okvira koji pruža programsku podršku kao uslugu (engl. *software as a service*), a to je .NET Framework.[17] Instalacija ovog okvira potrebna je za korištenje vrlo velikog broja Windows aplikacija jer su one zapravo i same razvijene u ovom programskom okviru. Arhitektura ovog okvira je vrlo je složena jer sadrži klasu sa velikim brojem biblioteka (FCL, engl. *Framework Class Library*) i pruža jezičnu interoperabilnost kroz nekoliko programskih jezika.

Programski okvir je jedinstveno programsko okruženje koje se najčešće veže uz jedan programski jezik i pruža mu određene nadogradnje za brži i bolji razvoj aplikacija, usluga i projekata općenito. Nadogradnje mogu uključivati prevoditelje, skupine biblioteka i sigurnosne module pa sve do raznih skupina alata i aplikacijskih programskih sučelja. U poglavlju 2. prikazane su neke od tehnologija na korisničkoj i poslužiteljskoj strani od kojih su većina bile programski okviri. Razlog tome je taj što se samo sa "izvornim" programskim jezikom ne može izraditi nekakva bolja tehnologija, dok se korištenjem programskih okvira vrlo brzo mogu razviti većina modernih aplikacija na poslužiteljskoj strani, uključujući web servise i web aplikacije. Kako navodi Wodehouse (2016.)[18], postoje dva tipa programskih okvira:

a) Okvir za razvoj stolnih aplikacija - okruženje koje se može iznova koristiti i najčešće je dio veće platforme (kao .NET Framework). Usmjereni su prema ubrzavanju razvoja programa i uključuju komponente kao što su biblioteke kodova, programi podrške, prevoditelji i aplikacijska programska sučelja za ubrzavanje toka podataka.

b) Okvir za razvoj web aplikacija - koriste se za usmjeravanje toka razvoja web aplikacija i web stranica. Najčešći oblik programskih okvira su MVC (engl. *model-view controller*) uzorka koji je nazvan prema tome što odvajaja logiku aplikacije u module (više o uzorcima i arhitekturi u potpoglavlju 5.2.)

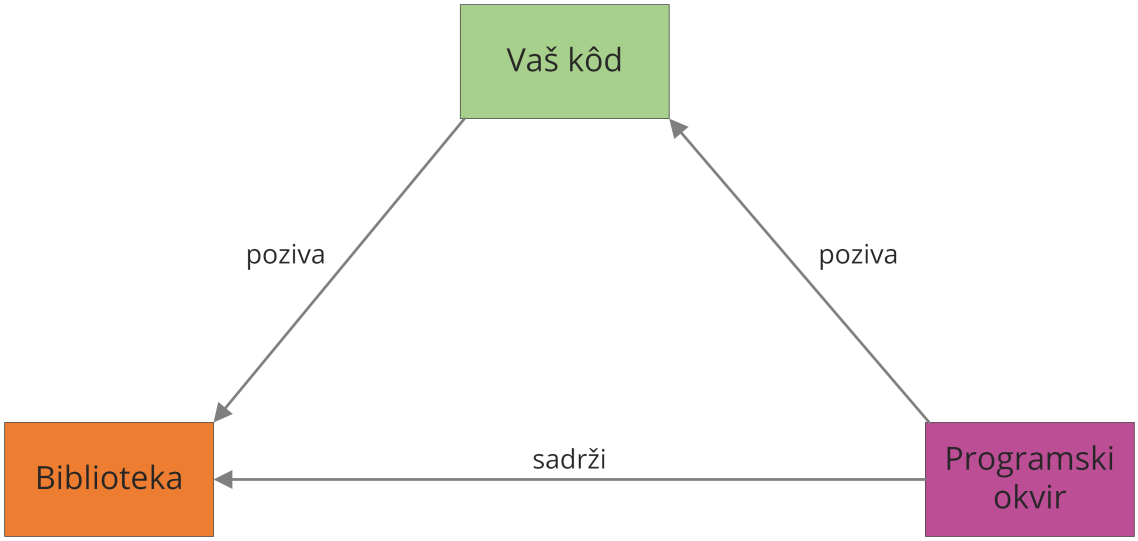
Na većini online izvora ne postoji distinkcija između programskog okvira i biblioteke. Tako naprimjer na jednom izvoru piše da je jQuery programski okvir, a na drugom biblioteka. Što je onda istinito? U ovom slučaju, jQuery je biblioteka jer je to izvorno samo jedna .js datoteka koja sadrži metode i objekte, što ju čini samo bibliotekom. Unatoč ovome, većina ljudi svejedno i dalje svrstava jQuery u JavaScript programske okvire. U teoriji, programski okvir je uvijek nešto veće od biblioteke jer najčešće i sam sadrži skup biblioteka. Međutim, postoje ključne raznolikosti koje odvajaju programske okvire od biblioteka, a Riehle (2000.)[19] je istaknuo najbitnije:

1) Inverzija kontrole - U okviru, za razliku od biblioteka ili sveobuhvatno programsko uprav-

ljanje tokom nije određeno od strane programera, već od strane programskog okvira

2) Mogućnost proširenja - Okvir može biti proširen od strane korisnika obično selektivnim obaranjem (engl. *override*) koda da se omoguće specifične funkcionalnosti

3) Nepromjenjivi kôd okvira: Okvir uvijek ima neko podrazumijevano ponašanje i stoga njegov kôd ne treba mjenjati, jedino proširiti po potrebi. Drugim riječima, korisnici mogu proširiti okvir, ali ne trebaju mjenjati njegov kôd.



Slika 10: *Programski okvir nasuprot biblioteke* (vlastita izrada)

Na slici 10. je zapravo prikazana spomenuta inverzija kontrole, odnosno kako programski okvir "diktira" tok programa na način da poziva kôd napisan od strane programera koji zatim poziva biblioteku.

Programski okviri ponekad mogu dodati veličini programa. Razlog tomu je taj što okvir bude razvijen za potrebe klijenta i u sebi sadrži brojna proširenja i funkcionalnosti. Stoga je potrebno prvo naučiti raditi s okvirom, a uloženo vrijeme u učenju okvira može koštati skoro kao i sam okvir. Uz ovo sve, potrebno je koristiti okvir i sve njegove mogućnosti u nadolazećim projektima kako bi se povratio uložen novac. Što se tiče odabira programskih okvira u razvoju web tehnologija, danas je najpopularniji MEAN tehnološki stog. MEAN je skraćenica za 4 tehnologije: Mongo DB, Express, Angular.js i Node.js. MongoDB je vrlo skalabilna noSQL baza podataka koja najbolje funkcionira sa Node.js poslužiteljskom platformom, Express je web programski okvir za Node.js i sa svojim brojnim proširenjima omogućava "ekspresni" razvoj web aplikacija i aplikacijskih programskih sučelja. Nadalje, Angular.js smatra se najboljim programskim okvirom na korisničkoj strani zbog ažurnog razvoja i testiranja aplikacije. Sve ove tehnologije u kombinaciji pružaju temelj i sve ono što je potrebno za razvoj modernih, brzih i skalabilnih aplikacija u skladu sa današnjih standardima na IT tržištu.

5.2. Programski okviri za JavaScript

U 3. poglavlju uviđeno je da upravljanje objektnim modelom dokumenta i slanje AJAX zahtjeva ponekad zna biti vrlo nezgrapno, pogotovo u razvoju nekih složenijih web aplikacija i u većim projektima. Isto tako, izvorni JavaScript ne pruža mogućnost kreiranja predložaka, popunjavanje koda, instantni ispis grešaka i drugih svojstava koje doprinose povećaju produktivnosti. Valja napomenuti i da su kod većih projekata učestali jedinični testovi, a testiranje bez korištenja ikakvog alata ili programskog okvira može biti vrlo dugotrajno i zamorno.

Svi prethodno navedeni razlozi su, uz razvoj web aplikacija i potreba tržišta, doveli do pojave prvih programskih okvira za JavaScript. Tako se 2005. godine pojavio jQuery kao DOM biblioteka koja olakšava pronalaženje, dohvaćanje i upravljanje elemenata objektnog modela. Ono što karakterizira ovu biblioteku i najviše odgovara programerima je upravljanje DOM-om pomoću CSS selektora, što omogućava jednostavno dohvaćanje DOM elemenata na isti način kao što se dohvaćaju HTML elementi u CSS-u.

Slika 11: *Evolucija JavaScript programskih okvira* [20]

Zatim je 2009. godine Google razvio okvir pod nazivom AngularJS. Privukao je pozornost korisnika zbog načina na koji proširuje HTML i učitava taj sadržaj za dinamički prikaz podataka. Danas je okvir poznat samo pod imenom Angular i to podrazumijeva najnovije izdanje (Angular 6), dok se AngularJS odnosni samo na prvobitnu verziju okvira. Još jedan programski okvir koji je imao veliki utjecaj na web razvoj je React. Razvijen je od strane Facebook tima 2013. godine i prvi je imao svojstvo kreiranja virtualnog DOM-a kojeg preglednik sprema u privremenu memoriju. Danas postoji oko 90 JavaScript programskih okvira i dijele se prema njihovoj namjeni:

a) Grafički okviri za vizualizaciju
b) Okviri za razvoj korisničkih sučelja
c) Okviri temeljeni na čistom JavaScriptu
d) Sustavi za predloške
e) Okviri za testiranje

f) Okviri orijentirani na web aplikacije (MVC, MVVM)

Smith (2018.)[21] navodi kako se od ostalih programskih okvira za razvoj korisničkih sučelja danas se najviše koriste: Vue.js, Meteor, Ember.js, Backbone.js, Aurelia.js i Polymer.

Jedna zanimljivost je da se prije nekoliko godina na StackOverflow stranici pojavilo pitanje u vezi misterioznog VanillaJS programskog okvira u smislu: Odakle sad ovaj okvir? Zna li netko nešto više o tome? Ova je tema privukla veliku pozornost s obzirom da je okvir bio veličine minimalnih 25 bajtova, što ga čini najlakšim okvirom. Naravno, poanta je bila u tome da se VanillaJS odnosi na izvorni JavaScript bez ikakvog programskog okvira, a dostupna datoteka ovog "okvira" je sadržavala samo obične JavaScript metode.

5.3. Opis i usporedba pojedinih programskih okvira

S obzirom da JavaScript programskih okvira ima previše da bi ih sve mogli detaljno opisati i usporediti, u nastavku će biti opisani samo "The Big Three" odnosno tri najveća okvira: Angular, React i Vue.js. Prethodno nabrojana tri okvira najviše su zastupljena na području web razvoja i imaju najveći broj suradnika na GitHub platformi.

Angular je definitivno najmoćniji programski okvir koji je dostupan za JavaScript. Točnije, Angular koristi TypeScript što je zapravo nadskup JavaScripta i primarno pruža kreiranje klasa i sučelja. To je odlična stvar za one koji se prebacuju iz objektno-orijentiranih jezika poput Java i C#. Još jedna prednost oko TypeScript dijalekta je to što obogaćuje okruženje time što ispisuje uobičajene greške dok korisnik piše kod. U Angular5 izdanju je poboljšana RxJS (Reactive Extensions JavaScript) mogućnost za još bolje upravljanje UI događajima i pozivima prema drugim programskim sučeljima. Također, Angular pruža dvostrano pridruživanje podataka (engl. *two-way data binding*) što osigurava očekivano ponašanje aplikacije te smanjuje rizik od pojavljivanja grešaka. Za razliku od ostala dva okvira, Angular koristi direktni DOM (kao u izvornom JavaScriptu), a ne virtualni DOM. Negativna stvar oko Angular okvira je ta što nova izdanja izlaze relativno često, a razlike između izdanja znaju vrlo odskakati. Tako naprimjer Angular5 koristi TypeScript 2.4 i prebacivanje sa Angular3 ili Angular4 na Angular5 može biti nezgodno. Poznate kompanije koje koriste Angular su: YouTube, Paypal, Nike, Google, Telegram, Freelancer, Udemy i mnogo drugih.

Za razliku od Angular-a koji se predstavlja kao poptuni programski okvir, React je samo JavaScript biblioteka za izgradnju korisničkoj sučelja. U tom cilju koristi komponente, a to su odvojeni dijelovi aplikacije koji se enkapsuliraju i moguće ih je koristiti u drugim dijelovima aplikacije. S obzirom da su enkapsulirani, ne mogu smetati višim dijelovima aplikacije. Još jedno karakteristično svojstvo ove biblioteke je JSX (JavaScript and XML) sintaksa koja omogućuje pisanje XML jezika unutar JavaScript-a. Može se zamisliti kao proširenje ECMAScript standarda, ali bez definirane sintakse. S obzirom da nije programski okvir, ne pruža puno dodatnih mogućnosti poput upravljanja putanjama (engl. *routing*) i upravaljanja stanjima (engl. *state*

management) pa za to mora koristiti druge vanjske tehnologije poput Redux biblioteke i React Router alata. Od sva tri navedena okvira, React je najlakši za naučiti i ima najviše dnevnih ažuriranja zahvaljujući suradnicima diljem svijeta. Ovo je u jednu ruku i negativna stvar jer upravo zbog ovakvih svakodnevnih ažuriranja teško je napraviti službenu dokumentaciju sa najnovijim informacijama. Iako je lagan za naučiti, može potrajati dugo vremena dok se usavrši jer React zahtjeva dublje znanje oko integracije korisničkog sučelja unutar MVC uzorka. Poznate kompanije koje koriste ovaj okvir su: Facebook, Instagram, Netflix, Yahoo, Whatsapp, Dropbox i druge.

Posljednji koji je izdan od "velike trojke" je Vue.js. Ovaj okvir je definitivno najlakši za naučiti i ima "najblažu" krivulju učenja. Tome doprinosi i detaljna dokumentacija na službenoj stranici koja omogućava početnicima da razviju neke aplikacije samo sa osnovnim znanjem HTML-a i JavaScript-a. Jedna bitna značajka ovog okvira je široka integracija, što znači da može biti korišten u razvoju jednostranih aplikacija, pa sve do složenih sustava. Negativna stvar je ta što ponekad implementacija okvira kao rješenja unutar postojećih većih sustava može uzrokovati problem. Isto tako, valja napomenuti da je autor ovog okvira Evan You i većina razvojnog tima čine Japanci i Kinezi. Zbog toga još uvijek postoji dio dokumentacije koji nije uopće (ili nije potpuno) preveden na engleski jezik.

6. Knockout.js

6.1. Općenito

Knockout je JavaScript biblioteka (a ne programski okvir) koju je 2010. godine objavio zaposlenik Microsoft kompanije Steve Anderson. Nastala je kao implementacija MVVM (engl. *model-view-viewmodel*) uzorka sa temeljnim ciljem odvajanja domene podataka, komponenata i prikaza tih podataka. Uz to, bio je fokus na kreiranju sloja koji će pružati jednoznačno upravljanje vezama između komponenti prikaza. Ono što čini ovu biblioteku jedinstvenom je ustrajnost u svojem temeljnom cilju, a ne imitacija drugih okvira. Stoga autor ove biblioteke i njegovi suradnici ne pokušavaju ugurati sve moguće funkcionalnosti unutar biblioteke već pratiti cilj, a to je povezivanje korisničkog sučelja sa ViewModel-om (Munro, 2015).

Ključni koncepti ove biblioteke su:

a) MVVM uzorak

b) Deklarativno povezivanje podataka (engl. *declarative binding*)

c) Automatsko osvježavanje korisničkog sučelja (kada se promjeni stanje modela podataka, korisničko sučelje se automatski ažurira)

d) Praćenje ovisnosti

e) Kreiranje predložaka (engl. *templating*)

Od ostalih karakteristika treba navesti da je Knockout besplatan, odnosno otvorenog kôda i koristi čisti JavaScript što znači da može biti integriran sa bilo kojim drugim web programskim okvirom. Veličine je sitnih 59kB jer izvorno ne sadrži ovisnosti (engl. *dependencies*). Ovisnosti se mogu shvatiti kao objekti koji su potrebni programu (biblioteci) kako bi ispravo funkcionirao. Svakako valja i napomenuti da Knockout podržavaju skoro pa svi web preglednici, čak i one starije verzije. Ukoliko želite koristiti sličnu tehnologiju na višem nivou, proučite Durandal - programski okvir koji koristi Knockout.

6.2. Instalacija

Instalacija Knockout-a kao i svake druge biblioteke vrši se preuzimanjem iste i uključivanjem u postojeću skriptu, što nije slučaj sa programskim okvirima koji zahtjevaju instalaciju isključivo preko npm-a (engl. *node package manager*). Stoga, instalaciju možemo izvršiti na jedan od sljedećih načina:

1. Uključivanje biblioteke sa CDN izvora - moguće je uključiti biblioteku koja je dostupna na CDN izvoru i samo ju referencirati unutar skripte:

```
<script
 ↪ src="http://ajax.aspnetcdn.com/ajax/knockout/knockout-3.4.2.js">
</script>
```

Ovaj način je definitivno najlakši jer ne zahtjeva preuzimanje biblioteke, ali može uzrokovati sporije učitavanje kôda jer biblioteka učitava sa vanjskog izvora i ne preporuča se u produkciji.

2. Preuzimanjem sa službene Knockout.js stranice - biblioteku je moguće preuzeti sa službene stranice

   http://knockoutjs.com/downloads/index.html

   i referencirati unutar skripte (na način objašnjem u poglavlju 3.3. Kako pisati JavaScript)

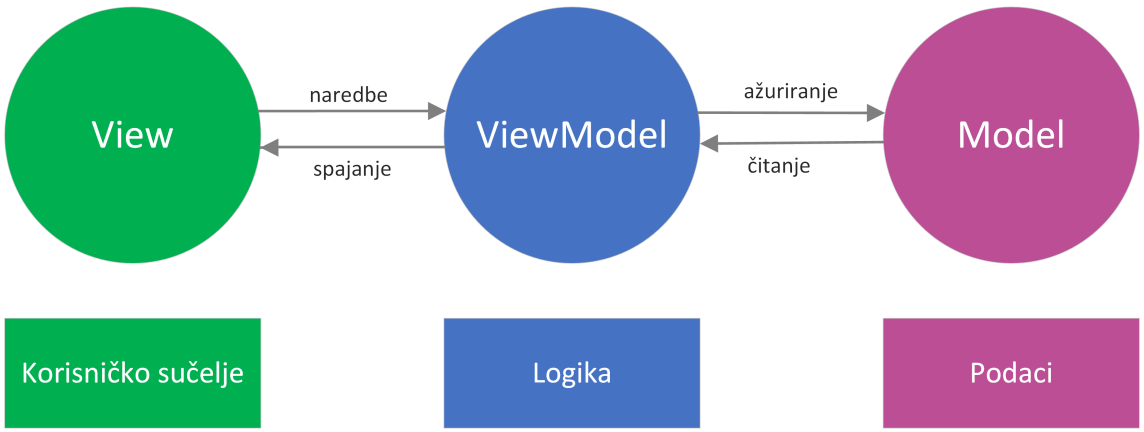
3. Preuzimanje putem upravitelja paketima (npm ili Bower) - unutar Node.js komandne linije potrebno je pozicionirati se unutar direktorija gdje želimo započeti novi projekt te unijeti sljedeću naredbu:

   ```
   npm install knockout
   ili
   bower install knockout
   ```

   To će preuzeti biblioteku i postaviti početno okruženje za korisnika.

6.3. MVVM uzorak

MVVM je uzorak dizajna ili arhitekture sustava koji se pretežito bazira na MVC (Model-View-Controller) uzorku, a sastoji se od 3 dijela: Model, View i ViewModel. U nastavku će se koristiti termini: Model, Srednji model i Prikaz.



Slika 12: *MVVM uzorak* (vlastita izrada)

Prvi princip ovog uzorka naziva se model (Model), a to može biti reprezentacija bilo kakvog objekta iz stvarnog svijeta.[22] Sukladno tome, mora sadržavati sva svojstva i metode koje bi opisivali taj objekt. Naprimjer, kada bi bio kreiran model koji bi predstavljao računalo, tada bi imao svojstva kao što su:

- NazivProizvođača (String)

- FrekvencijaJezgri (Number)
- DijagonalaZaslona (Number)

Također, imao bi metode poput:

- UpaliSe
- ReproducirajGlazbu

Koristeći Knockout, često ćete kreirati AJAX zahtjeve prema poslužiteljskoj strani koja će vraćati ovakve objekte JSON tipa.

Prikaz (View) je onaj vidljivi dio uzorka, odnosno interaktivni dio korisničkog sučelja koji prikazuje stanje promatrani modela. Isto tako šalje naredbe prema promatranom modelu (npr, kada korisnik klikne na gumb) i ažurira se kada dođe do promjena na istom. U ovom slučaju, prikazi su svi elementi sučelja, to jest HTML oznake koje opisuju sučelje (gumbovi, unosi, labele i sl.). Valja napomenuti da ovaj dio nema nikakvu logiku u pozadini.

Posljednji, i najbitniji dio ove arhitekture je srednji model (ViewModel). To je veza između modela i prikaza, odnosno programska reprezentacija podataka i operacija na korisničkom sučelju. A s obzirom da se nalazi između, srednji model sadrži nespremljene podatke s kojima korisnik trenutno radi.[23] Srednji model čine čisti JavaScript objekti, i oni nemaju nikakva znanja o HTML-u. Kada bi kreirali srednji model na temelju računala, on bi sadržavao sljedeća svojstva:

- PopisRacunala (Array)
- OdabranoRacunalo (Object)

Također, imao bi metode poput:

- DodajRacunalo
- IspisiRacunala

Nadalje, kako bi kreirali srednji model potrebno je samo instancirati JavaScript objekt:

```
var ViewModel = {};
```

Posljednji koncept MVVM uzorka je spajanje (engl. *binding*). Spajanje je ideja povezivanja svojstva i događaja na elementima korisničkog sučelja sa svojstvima i metodama objekta, odnosno promatranog modela. Naprimjer, spajanje bi se moralo izvršiti kada bi se htjela povezati gumb "Dodaj Računalo" na korisničkom sučelju sa metodom "DodajRacunalo" u prethodno navedenom promatranom modelu. Spajanje se izvršava naredbom:

```
ko.applyBindings(ViewModel)
```

Na ovaj se način zapravo "aktivira" Knockout i njegova najveća moć.

6.4. Deklarativno spajanje

U nastavku će biti kreiran prikaz koji će prikazivati podatake o nekom računalu:

```
<div>
    <h1>Racunalo</h1>
    <strong>Proizvodac: </strong>
    <span data-bind="text:racunalo.proizvodac"></span><br>
    <strong>Oznaka: </strong>
    <span data-bind="text:racunalo.oznaka"></span><br>
    <strong>Memorija: </strong>
    <span data-bind="text:racunalo.memorija"></span><br>
    <strong>Dijagonala zaslona: </strong>
    <span data-bind="text:racunalo.dijagonala"></span><br>
</div>
```

U JavaScript-u postoji globalni atribut `data-*` koji omogućuje pridruživanje posebnih svojstava na sve HTML elemente, a sastoji od prefiksa `data` i znaka `*` koji zamijenjuje proizvoljna oznaka. U Knockout-u je tako definiran atribut `data-bind` koji je temelj svakog prikaza u ovoj biblioteci jer se preko njega vrši spajanje elemenata iz prikaza i elementa (varijabli) iz srednjeg modela. Spajanje se sastoji od dvije stvari: naziva i vrijednosti koji su odvojeni dvotočkom. Takav način spajanja naziva se deklarativno spajanje, a u prethodnom prikazu korišten je 'text' atribut spajanja koji služi za tekstualni ispis vrijednosti. Na ovaj način moguće je u srednjem modelu mjenjati vrijednosti na sučelju pomoću varijabli koji odgovaraju imenu u prikazu. Osim spajanja za upravljanje tekstom i izgledom sučelja (text, html, visible, css itd.), postoje i spajanja za upravljanje tokom rada podataka (foreach, if, with itd.) te spajanja za rad sa obrascem i poljima za unos (value, click, event, enable itd.). Umjesto da se svakom elementu prikaza dodijeli jedinsteveni identifikator (id) i upravlja njime pomoću DOM-a, jednostavno mu se dodijeli bilo kakav JavaScript izraz (najčešće varijabla) koja će mjenjati njegovo stanje na sučelju. Knockout-ov sistem spajanja rezultira čišćom arhitekturom aplikacije, odnosno kodom i lakšim održavanjem. Valja svakako napomenuti da data-bind atribut inicijalno nema nikakvu funkciju jer web preglednik ne zna što znači, nego je potrebno napraviti `ko.applyBindings(vm)` metodu kako bi aktivirali spajanje (Ferrando, 2015.). Isto tako, potrebno je objekt 'racunalo' kreirati u odvojenoj .js datoteci. U ovom slučaju, datoteka će biti nazvana 'viewmodel.js'.

```
var vm =
{
    racunalo:
    {
        proizvodac: 'Dell',
        oznaka: 'Inspiron 3537',
        memorija: '8GB',
        dijagonala: 15
    }
```

```
};
ko.applyBindings(vm);
```

Kako bi se podaci iz srednjeg modela ispravno prikazali na sučelju, potrebno je uključiti promatrani model na kraju HTML-a odnosno nakon što se učita sav sadržaj:

```
<script src="js/viewmodel.js"></script>
```

Kada bi svaki put korisnik morao kreirati srednji model na prethodni način, kôd bi opet nakon nekog vremena postao nečitljiv. Stoga je poželjno kreirati JavaScript objekt koji sadrži sučelje računala, odnosno njegov model.

```
var Racunalo = function (proizvodac, oznaka, memorija,
 ↪ dijagonala)
{
    "use strict";
    var _proizvodac = proizvodac,
    _oznaka = oznaka,
    _memorija = memorija,
    _dijagonala = dijagonala;

    return
    {
        proizvodac = _proizvodac,
        oznaka = _oznaka,
        memorija = _memorija,
        dijagonala = _dijagonala
    };
};
```

Definiranje objekta koristeći ovu tehniku naziva se otkrivanje uzorka modula (engl. *revealing module pattern*) i omogućava jasno odvajanje javnih elemenata od onih privatnih. Ovaj objekt je također potrebno referencirati na kraju HTML dokumenta:

```
<script src="models/racunalo.js"></script>;
```

Sada je moguće iskoristiti ovaj model računala da se kreira srednji model računala na laški i elegantniji način:

```
var vm =
{
    racunalo: Racunalo ('Acer', 'Aspire 3', '4GB', 15)
};
ko.applyBindings(vm);
```

Ako se ponovno učita stranica u pregledniku, apsolutno ništa se neće izmijeniti. Međutim, sada je kreiranje promatranog modela lakše, a kôd je čitljiviji.

6.5. Automatsko osvježavanje sučelja

U prethodnim se primjerima nije u potpunosti prikazala dinamika na korisničkom sučelju. Zato u Knockout-u postoje svojstva koja se dinamički mjenjaju kroz korisničku interakciju, a ta se svojstva zovu promatrane vrijednosti (engl. *observables*). Promatrane vrijednosti su najmoćniji koncept ove biblioteke jer "obaviještavaju" program svaki put kada dođe do promjene na korisničkom sučelju, a realiziraju se sa `ko.observable` svojstvom.

```
var Racunalo = function (proizvodac, oznaka, memorija,
 ↪ dijagonala)
{
    "use strict";
    var _proizvodac = ko.observable(proizvodac),
    _oznaka = ko.observable(oznaka),
    _memorija = ko.observable(memorija),
    _dijagonala = ko.observable(dijagonala);

    return
    {
        proizvodac = _proizvodac,
        oznaka = _oznaka,
        memorija = _memorija,
        dijagonala = _dijagonala
    };
};
```

Sada će sve promjene na promatranim svojstvima u srednjem modelu biti automatski ažurirane na sučelju. Moguće je iznad prethodno kreiranog prikaza dodati sljedeći isječak HTML-a:

```
<div>
    <strong>Proizvodac: </strong>
    <input type="text" data-bind="value:racunalo.proizvodac"><br>
    <strong>Oznaka: </strong>
    <input type="text" data-bind="value:racunalo.oznaka"><br>
    <strong>Memorija: </strong>
    <input type="text" data-bind="value:racunalo.memorija"><br>
    <strong>Dijagonala zaslona: </strong>
    <input type="text" data-bind="value:racunalo.dijagonala"><br>
</div>
```

Ukoliko korisnik unese nove vrijednosti putem ovih polja za unos, te vrijednosti će odmah biti prikazane na sučelju. To je moguće zahvaljujući spajanjem pomoću 'value' atributa koji povezuje DOM element (input) sa svojstvom na srednjem modelu. Tek sada prava "magija" kod automatskog osvježavanja sučelja dolazi do izražaja. Svakako valja napomenuti da to NE ažurira vrijednosti u srednjem modelu, već ih samo privremeno sprema (više o tome u sljedećem

potpoglavlju). Ukoliko se želi samo jednu vrijednost dohvatiti ili privremeno promijeniti, to se može napraviti na sljedeći način:

```
racunalo.proizvodac(); // vraća "Dell"
racunalo.proizvodac("Lenovo"); // postavlja novu vrijednost
 ↪  "Lenovo"
```

6.6. Rad s podacima

U prethodnim se primjerima radilo sa statičkim podatcima koji su bili zapisani u srednji model. Međutim, razvoj bilo kakve aplikacije uvijek će zahtijevati dinamički rad s podacima, odnosno njihovo dohvaćanje i slanje prema poslužitelju. To se, kako je već objašnjeno u poglavlju 3.7. radi pomoću AJAX-a. S obzirom da Knockout ne pruža nekakve napredne opcije za rad sa poslužiteljskom stranom, za ovu svrhu će u sljedećim primjerima biti korištena jQuery biblioteka samo i isključivo za slanje AJAX zahtjeva.

U većini slučajeva će podatci s kojima se radi biti u JSON obliku, a struktura takve datoteke prikazana je u sljedećem primjeru:

```
{
"proizvodac" : "Toshiba",
"oznaka" : "Satellite L50",
"memorija" : "4GB",
"dijagonala" : 15
}
```

Datoteku je potrebno dohvatiti sa poslužitelja (ili lokalno) i srednji model popuniti s tim istim podatcima:

```
$.getJSON("json/Toshiba.json", function(data)
{
    vm.racunalo.proizvodac(data["proizvodac"]);
    vm.racunalo.oznaka(data["oznaka"]);
    vm.racunalo.memorija(data["memorija"]);
    vm.racunalo.dijagonala(data["dijagonala"])
});
ko.applyBindings(vm);
```

Na ovaj način svaki puta kada dođe do promjene na poslužitelju, isto tako će doći do promjene na sučelju i uvijek će biti dostupni ažurirani podatci. Za obrnuti postupak, odnosno za pretvorbu podataka iz srednjeg modela u čisti JSON objekt koristi se Knockout naredba `ko.toJSON`. Ona zapravo kreira JSON niz znakova koji predstavlja srednji model i u takvom obliku se može slati nazad na poslužitelj.

6.7. Upravljanje događajima

Za dodavanje prisluškivača događaja na neki HTML element, potrebno je postaviti "event" spajanje unutar njegove oznake. Tako je moguće i definirati rukovatelj događaja, odnosno JavaScript funkciju koja će se izvršiti kada se događaj "okine".

```
<div>
    <div data-bind="event: { mouseover: prikaziDetalje, mouseout:
     ↪ sakrijDetalje }">
        Prijeđi preko mene!
    </div>
    <p data-bind="visible: detaljiOtkriveni">Detalji teksta</p>
</div>
```

U prethodnom isječku koda je spojena funkcija `prikaziDetalje` sa događajem `mouseover`, i svaki puta kada se prijeđe mišem preko tog spremnika, izvršiti će se navedena funkcija. Isto vrijedi za događaj `sakrijDetalje` i funkciju `mouseout`. Primjetite i spajanje `visible` koje omogućava da odabrani element bude prikazan kada je istinit proslijeđeni parametar.

```
var vm = function () {
    this.detaljiOtkriveni = ko.observable(false);

    this.prikaziDetalje = function() {
        self.detaljiOtkriveni(true);
    };
    this.sakrijDetalje = function() {
        self.detaljiOtkriveni(false);
    }
};

var viewModel = new vm();
ko.applyBindings(viewModel);
```

Početna vrijednost varijable `detaljiOtkriveni` postavljena je na `false`. S obzirom da o ovoj varijabli ovisi vidljivost spremnika, potrebno je definirati funkcije `prikaziDetalje` i `sakrijDetalje` koje mjenjaju vrijednost te varijable, a okidaju se na događaje su navedeni u spajanju.

Primjetite da je ovaj put srednji model definiran kao funkcija, a ne kao objekt. Ovo korisniku daje nekoliko prednosti u odnosu na način koji je korišten u prošlim primjerima, a najbitniji je pristup do `this` ključne riječi. Ona je uvijek jednaka instanci srednjeg modela koji je kreiran pa se stoga može pisati `this.imeVarijable` umjesto puni naziv `viewModel.imeVarijable`. Isto tako, ako je srednji model definiran kao funkcija mogu mu se proslijediti parametri i postaviti početne vrijednosti promatranih varijabli jednake parametrima.

```
var ViewModel = function(ime, prezime) {
    this.Ime = ko.observable(ime);
    this.Prezime = ko.observable(prezime);
    this.Ispis = ko.computed(function() {
        return this.ime() + " " + this.prezime();
    }, this);
};
```

Nadalje, korisniku se daje nekakva sloboda jer ne mora sve varijable i funkcije definirati kao **naziv : vrijednost** i odvajati zarezom što može uzrokovati puno sintaktičkih greški. Valja napomenuti da ako se srednji model definira na ovaj način, potrebno je prvo kreirati instancu modela i onda tek pozvati funkciju `ko.applyBindings()` nad prethodno kreiranom instancom.

7. Ogledna aplikacija

7.1. Opis aplikacije

U sljedećih nekoliko potpoglavlja biti će opisan razvoj ogledne aplikacije pod nazivom Cultural Uplift. Ideja je da se izradi web aplikacija za kupovinu karata za glazbeno-kulturne festivale na području cijele Hrvatske. Aplikacija će sadržavati osnovne karakteristike svake web aplikacije kao što su registracija, prijava, pregled i kupovina karata, odjava i sl. Korisnik će se na stranici moći registrirati, pretraživati, pregledavati i kupovati ulaznice te se moći dobiti kratke informacije o svakom pojedinom festivalu. Osim običnog korisnika, na aplikaciji će postojati još dva tipa korisnika, a to su administrator i voditelj festivala. Administrator kreira festivale i dodjeljuje voditelje svakom festivalu te ima pristup pregledu statistike festivala u obliku raznih grafova. S obzirom da Knockout.js i čisti JavaScript ne pružaju puno mogućnosti oko vizualizacije podataka, u ovu svrhu biti će korištena Chart.js biblioteka za jednostavno kreiranje i prikazivanje raznih oblika grafova. Nadalje, voditelj kreira područja za festival koji vodi te određuje naziv, cijenu i raspoloživu količinu ulaznica za to područje. Aplikacija će raditi na Apache web poslužitelju i svi podaci s kojima aplikacija radi će biti pohranjeni u MySQL bazu podataka, a kao jezik na poslužiteljskoj strani biti će korišten PHP. Što se tiče korisničke strane, koristit će se Knockout.js biblioteka za razvoj interaktivnog korisničkog sučelja i jQuery biblioteka isključivo za pisanje AJAX poziva. Uz to, prilikom izrade svakog prikaza aplikacije biti će korištene već gotove komponente iz jednostavne HTML/CSS/JS biblioteke Bootstrap koja će olaškati i ubrzati razvoj korisničke strane te će ovoj aplikaciji dati izgled poput današnjih modernih aplikacija.

7.2. Poslužiteljska strana

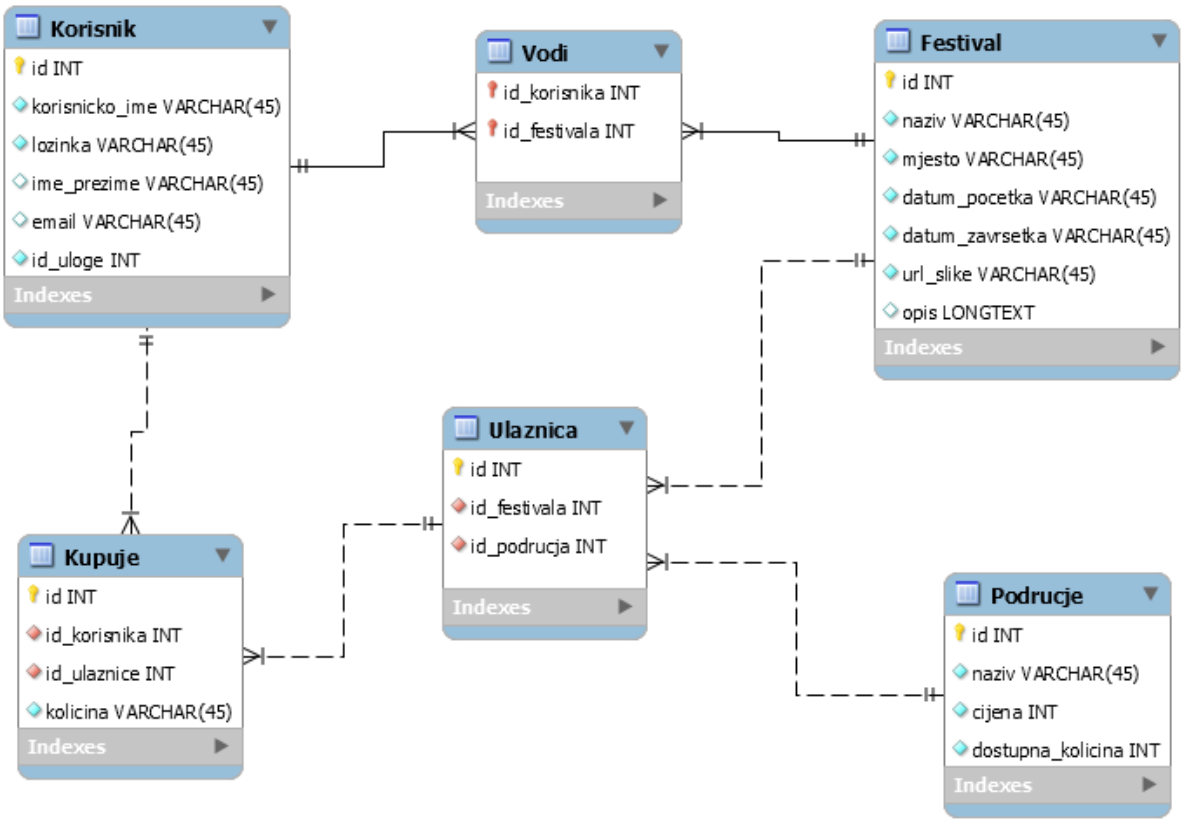
Prije implementacije aplikacije na postojeći poslužitelj (hosting), sav razvoj se radio lokalno pomoću XAMPP distribucijskog paketa koji uključuje instalaciju Apache lokalnog poslužitelja, MySQL baze podataka, phpMyAdmin alata za administraciju i modeliranje baze podataka te PHP jezik. Uz pomoć ovog alata podešeno je razvojno okruženje poslužiteljske strane sa lokalnim poslužiteljem i bazom podataka te je omogućeno lako testiranje aplikaciju nakon svake promjene umjesto da se ažuirane datoteke prvo šalju na udaljeni poslužitelj i tek onda testiraju. Podaci iz baze podataka kasnije se mogu izvesti i lagano uvesti u drugu MySQL bazu podataka.

7.2.1. ERA model

ERA model je metoda konceptualnog i fizičkog modeliranja podataka. Osnovni koncept ovog modela su entiteti, atributi, veze i ograničenja. Ovaj model (Slika 12.) izrađen je za aplikacijski sustav i služi za evidenciju korisnika i festivala, te za prikaz veza između entiteta u bazi podataka. Gore navedeni model ima 5 jakih entiteta (Korisnik, Festival, Kupuje, Ulaznica i Područje) koji mogu stajati zasebno i ne ovise o drugim entitetima, te jedan slabi entitet Vodi jer povezuje entitete u M:N vezi pa između njih mora stajati slabi entitet. Iznimka su entiteti Kupuje i Ulaznica jer se želi do svake narudžbe korisnika i ulaznice doći jednoznačno, pa mora postojati

identifikator za svaki red.

Ovih 6 entiteta predstavlja tablice u bazi podataka Cultural Uplift aplikacije u koje se spremaju svi podaci potrebni za rad aplikacije i ispunjenje poslovnih zahtjeva i logike. Kroz entitet Korisnik biljeze se svi korisnicki racuni, a za novi red potrebno je unijeti korisnicko ime i lozinku i ID uloge koji se inicijalno postavlja na 1 za običnog korisnika (2 je voditelj, a 3 administrator). U tablicu Festival upisuju se svi festivali sa mjestom i vremenom održavanja i kratkim opisom, a atribut url_slike sadrži relativnu putanju do učitane slike. S obzirom da festival može imati više područja (VIP, tribina, stajanje i sl.), ovdje se radi o M:N vezi i između se nalazi entitet Ulaznica koji ih povezuje. Za područje je potrebno unijeti naziv, cijenu i raspoloživu količinu ulaznica. Sve narudžbe korisnika spremaju se u tablicu Kupuje, a na razini tablice potrebno je postaviti okidač 'UmanjiDostupnuKoličinu' koji će smanjiti dostupnu količinu ulaznica za određeno područje svaki put kad je unesen red u tablicu Kupuje sa identifikatorom ulaznice koji se odnosi na to područje.



Slika 13: *ERA model* (vlastita izrada)

7.2.2. Popis i opis skripata

Skripte na poslužiteljskoj strani aplikacije su podijeljene u nekoliko skupina radi lakše snalažljivosti i organizacije toka rada.

Skripte stranica predstavljaju stvarne fizičke stranice na kojima se nalazi sadržaj i stoga ih nije potrebno dodatno opisivati, a razlog zašto se ovdje ne radi o običnim HTML datotekama je taj što u svakoj stranici uključujemo 'aplikacijski-okvir.php' skriptu i pozivamo metode iz ostalih skripti.

- index.php
- registracija.php
- pocetna.php
- kupi-ulaznicu.php
- pregled-statistike.php
- pregled-ulaznica.php
- upravljanje-ulaznica.php
- upravljanje-voditeljima.php

Pomoćne skripte nemaju nikakav sadržaj, ali sadrže brojne operacije za generiranje sadržaja i preusmjeravanje među skriptama.

- aplikacijski-okvir.php - skripta koja uključuje sve ostale skripte i klase, potrebno ju je uključiti (engl. *include*) u svaku skriptu stranica
- kreiranje-festivala.php - skripta koja prima POST parametre iz forme za kreiranje festivala i unosi ih u bazu podataka, a izdvojena je iz razloga što provjerava i prenosi sliku festivala na udaljeni poslužitelj
- login.php - pomoćna skripta koja autenticira korisnika i kreira sesiju ako se radi o uspješnoj autentikaciji te nakon toga prijavljuje korisnika i preusmjerava na početnu stranicu, u suprotnom ispisije poruku o neuspješnoj autentikaciji
- logout.php - pomoćna skripta koja odjavljuje korisnika, odnosno briše sesiju i preusmjerava na index.php stranicu
- obrada-zahtjeva.php - najvažnija pomoćna skripta koja obrađuje sve AJAX zahtjeve koji su došli sa korisničke strane. Prema skripti se šalje ključna riječ i parametri ako su potrebni, a skripta provjerava ključnu riječ i ovisno o rome radi određenu akciju i vraća odgovor (provjerava dostupno korisničko ime, dohvaća festival, kreira ulaznicu i sl.)

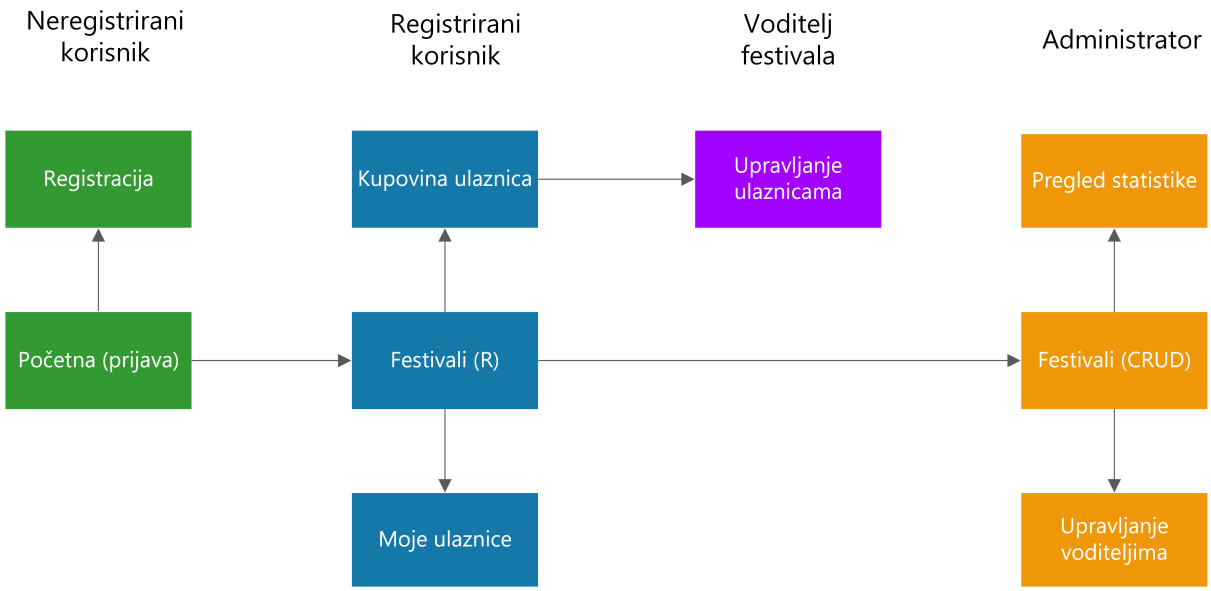
Klase su izdvojene cjeline koje predstavljaju predloške objekta, a sadrže posebne operacije za promjene stanja svojstva unutar te klase.

- baza.class.php - klasa čija se instanca kreira samo kod rada s bazom podataka, te se pozivaju metode za spajanje na bazu, izvršavanje upita i zatvaranje veze (preuzeto iz nastavnih materijala kolegija Web Dizajn i Programiranje)
- funkcije.class.php - posebna klasa koju nije potrebno instancirati, a sadrži samo specifične metode koje se pozivaju u skriptama stranica (postavljanje HTTPS-a, generiranje HTML sadržaja i sl.)

- sesija.class.php - klasa koja sadrži metode za kreiranje sesije, dohvaćanje vrijednosti iz sesije i brisanje sesije (preuzeto iz nastavnih materijala kolegija Web Dizajn i Programiranje)

7.2.3. Navigacijski dijagram

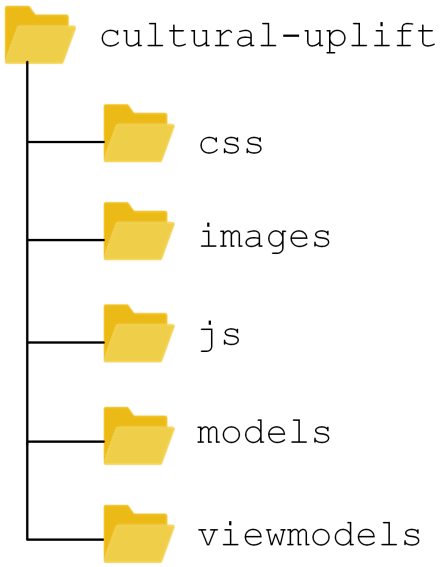
Sljedeći navigacijski dijagram jasno prikazuje navigaciju kroz aplikaciju i lokacije koje su dostupne svakom tipu korisnika. Neregistrirani korisnik može pristupiti samo početnoj stranici sa prijavom i registraciji. Nakon registracije, on postaje registrirani korisnik i može pregledavati festivale, kupovati ulaznice za festivale i pregledavati kupljene ulaznice. Ako je registriranom korisniku dodijeljena uloga, on postaje voditelj festivala i može kreirati i uređivati područja za taj festival. Uz sve prethodno navedeno, administrator može još kreirati festivale te dodijeljivati voditelje festivalima i pregledavati statistiku festivala.



Slika 14: *Navigacijski dijagram* (vlastita izrada)

7.3. Korisnička strana

Prije početka razvoja korisničke strane, potrebno je postaviti pravilnu strukturu direktorija u kojem se nalazi projekt. U mapi `css` nalaze se sve stilske upute uključujući i one iz Bootstrap biblioteke. JavaScript biblioteke (jQuery i Knockout) nalaze se u `js` direktoriju, a svi srednji modeli nalaziti će se u `viewmodels` direktoriju. Važno je napomenuti da će za svaku stranicu (koja to zahtjeva) biti izrađen zaseban srednji model jer tako funkcionira arhitektura Knockout biblioteke, a svakako će učiniti kod čitljivijim. Svi poslovni modeli se nalaze u direktoriju `models`, a sve slike koje aplikacija koristi i koje se prenesu na poslužitelj nalaze se u direktoriju `images`. Konačna struktura direktorija trebala bi izgledati ovako:



Slika 15: *Struktura direktorija projekta* (vlastita izrada)

7.3.1. Registracija i validacija obrasca

Prilikom razvoja svake funkcionalnosti prvo je potrebno kreirati prikaz, odnosno HTML sadržaj stranice. U ovom slučaju, prvo se kreira forma za registraciju i radi deklarativno spajanje elemenata koje će biti potrebno validirati tijekom unosa. Iz Bootstrap biblioteke preuzeta je `form-group` komponenta za unos podataka, koja je zapravo spremnik (engl. *container*) i u sebi sadržava polje za unos i oznaku elementa koji su prilagodljivi spremniku u kojem se nalaze.

```
<div class="form-group">
    <label for="input-email"><strong>E-mail adresa</strong></label>
    <input data-bind="value: Email, valueUpdate: 'afterkeydown',
     ↪ event:
    { 'keyup' : provjeriIspravnostEmail }" type="text"
 ↪ class="form-control" id="input-email"
 ↪ placeholder="peroperic@foi.hr">
    <small data-bind="visible: NeispravanEmail" class="form-text
     ↪ text-muted">E-mail nije u pravilnom formatu!</small>
</div>
```

Vrijednost elementa za unos je spojena pomoću `value` spajanja i postavljene su opcije `valueUpdate: 'afterkeydown'` pomoću koje će se vrijednost varijable u srednjem modelu ažurirati svaki put kada korisnik pristisne tipku, te opcija `event:{ 'keyup': provjeriIspravnostEmail}` koja će na svaki pritisak tipke korisnika izvršavati funkciju za provjeru da li je uneseni e-mail u pravilnom formatu. Ispod polja za unosa nalazi se upozorenje o nepravilnom formatu e-maila koje se prikazuje samo kada je svojstvo `NeispravanEmail` istinito, a to je realizirano pomoću `visible` svojstva.

```
self.Email = ko.observable();
self.NeispravanEmail = ko.observable(false);
```

```
self.provjeriIspravnostEmail = function() {
    var reg = /(\w\.?)+@(\w+\.[^.])+(\w{1,})/;
    var regex = RegExp(reg, "");

    if (regex.test(self.Email())) {
            self.NeispravanEmail(false);
    }
    else {
            self.NeispravanEmail(true);
    }
}
```

Početna vrijednost svojstva `NeispravanEmail` postavljena je na `false`, a svaki put kada unos ne prođe testiranje izraza, vratiti će se na `true` vrijednost i poruka o upozorenju će se prikazati.

Na polje za unos korisničkog imena je osim vrijednosti i deklarativno spojen osluškivač događaja `focusout` koji će izvršiti funkciju `provjeriKorisničkoIme` svaki puta kada je maknut fokus sa polja za unos korisničkog imena. Ispod polja se također nalazi upozorenje o zauzetom korisničkom imenu koje se prikazuje kada je svojstvo `NedostupnoKorisnickoIme` istinito.

```
<div class="form-group">
    <label for="input-username"><strong>Korisnicko
     ↪ ime</strong></label>
    <input data-bind="value: Korisnicko_ime, event: { focusout :
     ↪ provjeriKorisnickoIme }" class="form-control" type="text"
     ↪ id="input-username" placeholder="peroperic">
    <small data-bind="visible: NedostupnoKorisnickoIme"
     ↪ class="form-text text-muted">Korisnicko ime je
     ↪ zauzeto!</small>
</div>
```

Funkcija `provjeriKorisnickoIme` šalje AJAX zahtjev sa kljucnom riječi `provjeri-korisnicko-ime` i unesenim korisničkim imenom prema pomoćnoj skripti `obrada-zahtjeva.php` koja provjerava da li u bazi već postoji korisnik sa tim korisničkim imenom. Ukoliko postoji, skripta će vratiti "Nedostupno" i na sučelju će se prikazati upozorenje jer će svojstvo `NedostupnoKorisnickoIme` postaviti na `true`. Ova provjera je mogla biti realizirana i na pritisak tipke kao što je slučaj sa e-mail poljem, ali bi slanje desetak AJAX zahtjeva u tri sekunde dovelo do zamrzavanja sučelja.

```
self.Korisnicko_ime = ko.observable();
self.NedostupnoKorisnickoIme = ko.observable(false);
```

```
self.provjeriKorisnickoIme = function() {
        if (self.Korisnicko_ime() != null) {
        $.ajax({
            url: 'obrada-zahtjeva.php',
            method: 'POST',
            dataType: 'text',
            data: {
                key: 'provjeri-korisnicko-ime',
                korisnicko_ime: self.Korisnicko_ime()
                }, success: function (response) {
                        if ($.trim(response) == "Dostupno") {
                                self.NedostupnoKorisnickoIme(false);
                        }
                        if ($.trim(response) == "Nedostupno") {
                                self.NedostupnoKorisnickoIme(true);
                        }
                }, error: function (response) {
                    console.log(response);
                }
            })
        }
}
```

Polje za unos lozinke se također provjerava nakon micanja fokusa te je uz to deklarativno spojena i `css` klasa `neispravan_unos` koja će se primjeniti svaki puta kada svojstvo `NeispravnaLozinka` bude istinito. Ta klasa će postaviti polju za unos crveni obrub i dati korisniku do znanja da je lozinka u neispravnom formatu.

```
<div class="form-group">
    <label for="input-password"><strong>Lozinka</strong></label>
    <input data-bind="value: Lozinka, event: { blur: provjeriLozinku
     ↪ },
    css: { neispravan_unos : NeispravnaLozinka() == true }"
 ↪ type="password" class="form-control"
 ↪ aria-describedby="passwordHelp" id="input-password"
 ↪ placeholder="Lozinka">
    <small id="passwordHelp" class="form-text text-muted">Lozinka
     ↪ mora sadržavati 6-10 znakova.</small>
</div>
```

Funkcija `provjeriLozinku` samo provjerava da li je duljina unesene lozinka između 6 i 10 znakova, te ovisno o ispunjenosti uvjeta postavlja vrijednost svojstva `NeispravnaLozinka`. Konačno, kada korisnik stisne na gumb "Registriraj se", izvršava se funkcija `registrirajKorisnika` koja provjerava da li su unesena sva polja i šalje podatke prema skripti `obrada-zahtjeva.php`

koja upisuje novog korisnika u bazu podataka. Kada skripta vrati odgovor o uspjehu, preusmjerava se na stranicu za prijavu i prikazuje se poruka o uspješnoj registraciji.

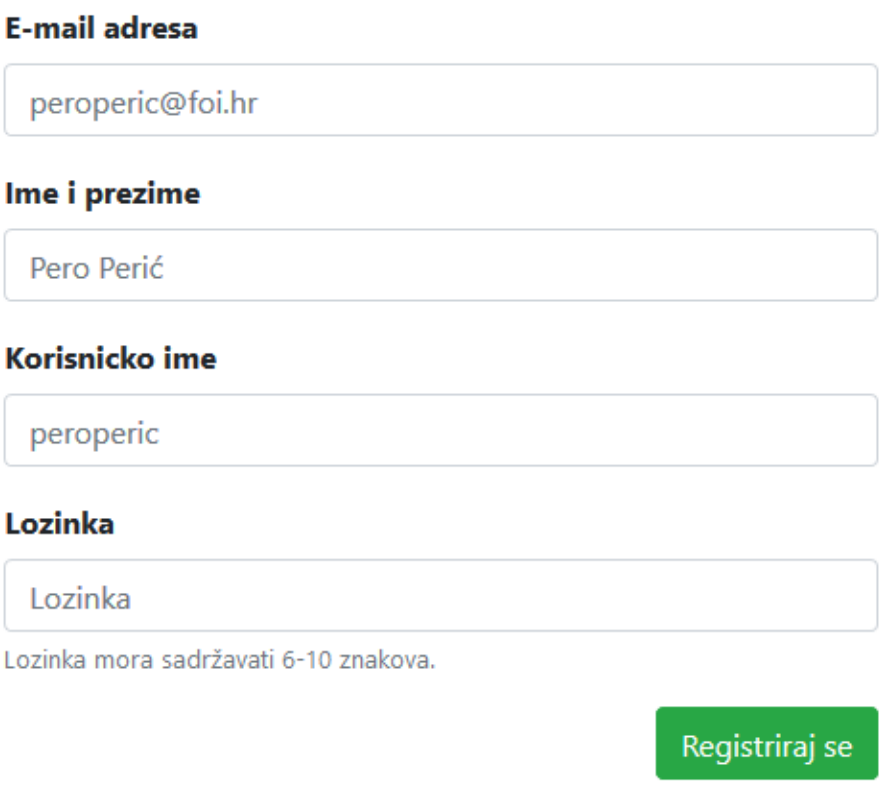
```
self.registrirajKorisnika = function(formElement) {
    if (self.provjeriSvaPolja() == true) {
    $.ajax({
        url: 'obrada-zahtjeva.php',
        method: 'POST',
        dataType: 'text',
        data: {
            key: 'registriraj-korisnika',
            email: self.Email(),
            ime_prezime: self.ImePrezime(),
            korisnicko_ime: self.Korisnicko_ime(),
            lozinka: self.Lozinka()
        }, success: function (response) {

          ↪ window.location.assign("index.php?mod=uspjesnaRegistracija");
        }, error: function (response) {
            console.log(response);
        }
    })
    }
}
```

Treba svakako napomenuti da je sadržaj svake stranice postavljen kao prilagodljivi spremnik unutar kojeg se stavljaju svi ostali spremnici sa sadržajem. To je napravljeno uz pomoć CSS opcije `grid` koja kreira nešto poput koordinatne mreže na definiranom spremniku i onda je moguće sve elemente jednostavno postavljati unutar mreže, gdje god korisnik želi.

```
.form-registration {
    display: grid;
    grid-template-columns: 2fr 2fr 2fr;
    grid-template-rows: auto;
}

.form-registration-inputs {
    margin-top: 3em;
    grid-column: 2/3;
    grid-row: 1/2;
}
```

U prethodnom isječku koda kreirana je mreža sa 3 stupca, od kojih su svaki širine 2 segmenta ekrana (2fr) te proizvoljnim brojem redaka. Nakon toga je postavljena forma sa poljima za unos u stupac 2-3 (srednji stupac) i prvi red. Ovo omogućava da forma uvijek zauzima 2 segmenta ekrana, neovisno o veličini zaslona i uvijek je bolje koristiti ovakav pristup umjesto statičnih

spremnika sa fiksnim dimenzijama. Konačno sučelje obrasca za registraciju trebalo bi izgledati ovako:



Slika 16: *Forma za registraciju* (vlastita izrada)

7.3.2. Ispis festivala

Ispis festivala na početnoj stranici izvršava se pozivom statičke funkcije `IspisiFestivale` unutar klase `funkcije.class.php`. Razlog zašto se radi o statičkoj funkciji je taj jer se za ispis festivala ne treba slati nikakve parametre, odnosno uvijek će se ispisati svi festivali bez obzira na sve. Funkcija dohvaća sve festivale iz baze podataka, iterira kroz svaki zapis te generira HTML sadržaj (Kartu) i popunjava podacima tog festivala.

```
<div class='card text-center'>
    <img class='card-img-top' src='$url_slike'>
    <div class='card-body'>
        <h5 class='card-title'>$naziv</h5>
        <p class='card-text'>$mjesto</p>
        <p class='card-text'>$datum_pocetka - $datum_zavrsetka</p>
        <a href='kupi-ulaznicu.php?id=$id' class='btn
         ↪ btn-primary'>Pregled</a>
    </div>
</div>
```

Svaki festival se ispisuje u obliku karte (vidi Slika 17.), a ova komponenta je preuzeta iz Bootstrap biblioteke jer je savršena za ispis pojedinog festivala zbog sličice iznad i kratkog opisa sa gumbom ispod.

Slika 17: *Prikaz festivala u obliku karte* (vlastita izrada)

7.3.3. Kupnja ulaznica

Kada korisnik klikne na festival, na isti način kao što je navedeno u prošlom odlomku ispisat će se sve informacije o tom festivalu. Ispod se nalazi tablica sa zaglavljem koja predstavlja košaricu sa stavkama i to je zapravo najsloženiji dio aplikacije jer zahtjeva korištenje predložaka, izračunatih vrijednosti (engl. *computed values*) i rad sa poljima promatranih vrijednosti (engl. *observable arrays*). Košarica je inicijalno prazna, a korisnik dodaje nove stavke klikom na gumb "Dodaj ulaznicu". S obzirom da će se svaki red popunjavati s podacima, potrebno je kreirati Knockout predložak koji će sadržavati promatrane vrijednosti:

```
<script type="text/html" id="stavkaPredlozak">
    <tr>
        <td><select class="form-control" data-bind='options:
        ↪ ulaznice,
          optionsText: "naziv",
          value: "id",
          optionsCaption:"--Odaberi--",
          value: podrucje'>
          </select>
        </td>
        <td data-bind='with: podrucje'>
```

```
            <span data-bind='text: cijena' ></span>
        </td>
        <td data-bind='with: podrucje'>
            <span data-bind='text: dostupna_kolicina'></span>
        </td>
        <td>
            <input class="form-control" data-bind='value: kolicina,
             ↪ valueUpdate: "afterkeydown"' />
        </td>
        <td>
            <span data-bind='text:
             ↪ formatirajIspis(cijenaStavke())'></span>
        </td>
        <td>
            <button class="btn btn-danger" data-bind='click:
             ↪ $parent.ukloniStavku'>Ukloni</button>
        </td>
    </tr>
</script>
```

Prethodni isječak koda zapravo predstavlja red koji će se svaki put dodavati u tijelo tablice kada se doda nova stavka u košaricu, a također je potrebno kreirati JavaScript objekt toga reda:

```
var stavkaNarudzbe = function ()
{
    var self = this;
    self.ulaznice = ko.observableArray(_ulaznice);
    self.podrucje = ko.observable(1);
    self.cijena = ko.observable(1);
    self.kolicina = ko.observable(1);
    self.cijenaStavke = ko.computed(function ()
    {
        return self.podrucje() ? self.podrucje().cijena *
         ↪ parseInt("0" + self.kolicina(), 10) : 0;
    });

    self.podrucje.subscribe(function()
    {
        self.kolicina(1);
    });
};
```

Naravno, aplikacija ne radi sa statičkim podacima nego dinamičkim podacima u bazi podataka. Stoga je potrebno kreirati funkciju koja se izvršava na početku skripte koja šalje AJAX zahtjev prema pomoćnoj skripti koja kao odgovor vraća sve ulaznice, odnosno područja tog festivala u JSON obliku i upisuje ih u globalno polje `_ulaznice`. Zbog toga je moguće deklarativno

spojiti element `select` i kao opcije postaviti promatrano polje ulaznica koje se prethodno dohvatilo. Sukladno tome, svaka opcija će zapravo biti jedan red polja koji sadrži atribute `id`, `naziv`, `cijena` i `dostupna_kolicina` i moguće je cijelu opciju deklarativno spojiti kao promatranu vrijednost `podrucje`. Kada korisnik odabere opciju (podrucje), automatski će se prikazati cijena i dostupna količina ulaznica tog područja zbog spajanja `with: podrucje`. U predzadnjem stupcu reda nalazi se cijena stavke koja predstavlja izračunatu vrijednost i množi cijenu područja sa unesenom količinom i ispisuje vrijednost na sučelju. Posljednja funkcija `subscribe` služi samo da se količina automatski postavi na 1 čim korisnik odabere opciju, odnosno čim se "pretplati" (engl. *subscribe*).

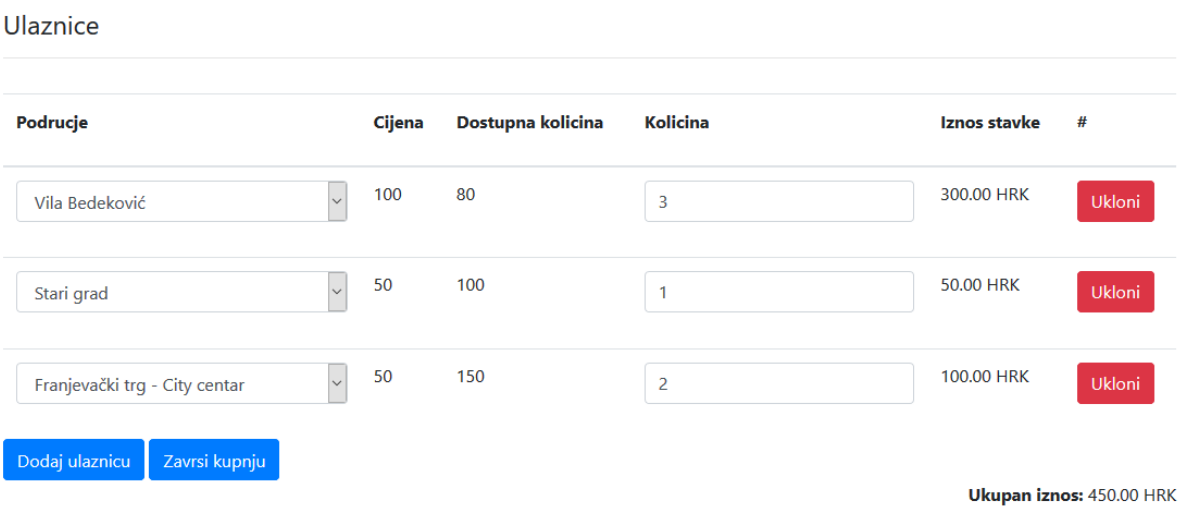
```
var Kosarica = function () {
    var self = this;
    self.stavke = ko.observableArray([]);
    self.ukupanIznos = ko.pureComputed(function() {
        var total = 0;
        $.each(self.stavke(), function() { total +=
         ↪ this.cijenaStavke() })
        return total;
    });

    self.dodajStavku = function() { self.stavke.push(new
     ↪ stavkaNarudzbe()) };
    self.ukloniStavku = function(stavka) { self.stavke.remove(stavka)
     ↪ };
    self.zavrsiNarudzbu = function() {
        var narudzbaPodaci = $.map(self.stavke(), function(stavka) {
            return stavka.podrucje() ? {
            nazivUlaznice: stavka.podrucje().naziv,
            idUlaznice: stavka.podrucje().id,
            kolicina: stavka.kolicina()
                } : undefined
            });

            $.ajax({
                    url: 'obrada-zahtjeva.php',
                    method: 'POST',
                    dataType: 'text',
                    data: {
                        key: 'izvrsi-narudzbu',
                        narudzba: narudzbaPodaci
                    }, success: function (response) {

                         ↪ window.location.assign('pregled-ulaznica.php');
                    }, error: function (response) {
                        console.log(response);
                    }
```

```
                    })
    };
};
```

Kosarica je zapravo srednji model koji u sebi sadrži polje promatranih vrijednosti, odnosno stavki narudzbe. Već je rečeno da je kosarica inicijalno prazna, a deklarativno je spojena sa prethodno navedenim poljem pomoću `foreach: stavke` opcije. Sljedeći isječak koda iterira kroz sve polje sa stavkama narudžbe i dodaje red u tablicu prema deklariranom predlošku *stavkaPredlozak* i popunjava ga podacima prema predlošku.

```
<tbody data-bind='template: {name: "stavkaPredlozak", foreach:
 ↪  stavke}'></tbody>
```

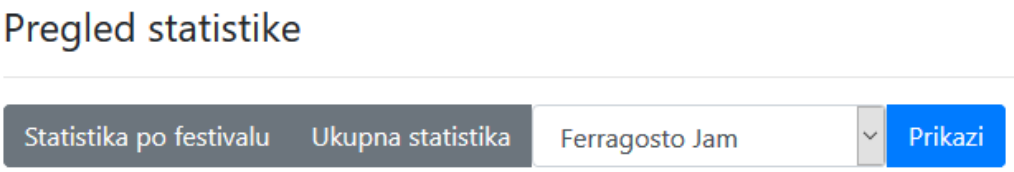
Nadalje, srednji model sadrži metode `dodajStavku` koja jednostavno dodaje jedan element u polje sa stavkama pa tako i novi red u tablici, te metodu `ukloniStavku` koja je je deklarativno spojena sa gumbom još u predlošku reda pomoću `$parent.ukloniStavku`. Svakako valja napomenuti da je `$parent` notaciju moguće koristiti isključivo ako se element nalazi unutar područja u kojem je definirana metoda. Najvažnija metoda `zavrsiNarudzbu` iterira kroz sve stavke narudžbe i sve atribute i vrijednosti upisuje u varijablu `narudzbaPodaci` u JSON obliku. Ta je varijabla odmah pogodna za slanje AJAX zahtjevom prema pomoćnoj skripti koja narudžbu upisuje u bazu podataka. Konačni prikaz košarice trebao bi izgledati ovako:



Slika 18: *Košarica sa ulaznicama* (vlastita izrada)

7.3.4. Pregled statistike

Crtanje grafova za prikaz statistike pojedinog festivala je realizirano uz pomoć Chart.js biblioteke. Na sučelju se nalazi navigacijska traka na kojoj korisnik može odabrati između ukupne statistike u kojoj mu se prikazuje sveukupna zarada svakog festivala i broj prodanih ulaznica svakog festivala ili statistike svakog pojedinog festivala.



Slika 19: *Navigacijska traka kod statistike* (vlastita izrada)

Prvo je potrebno kreirati dva elementa `canvas` koji će inicijalno biti prazni i neće zauzimati prostora na sučelju, a u njih će se crtati grafovi.

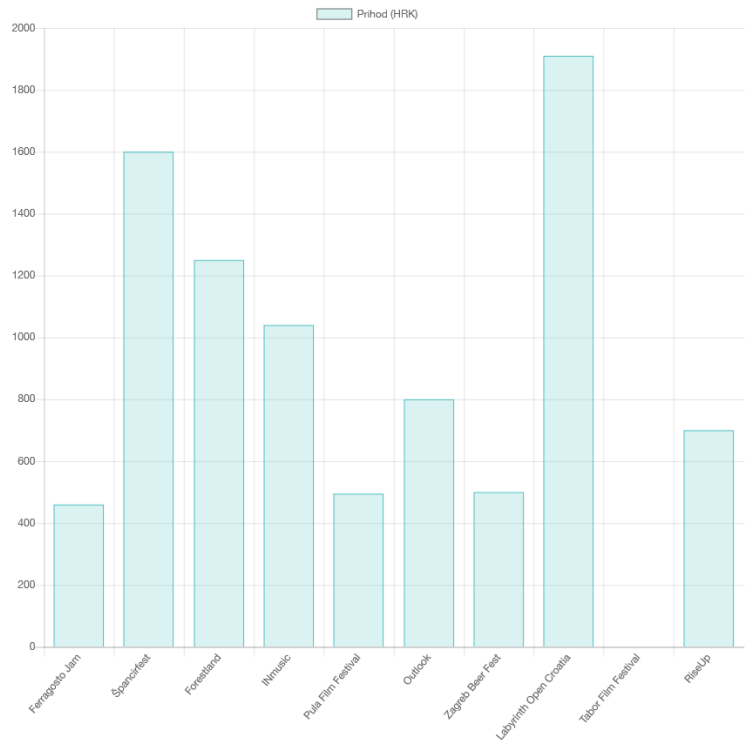
```
<canvas id="grafPrihod" width="400" height="400"></canvas>
<canvas id="grafBrojUlaznica" width="400" height="400"></canvas>
```

Ako korisnik odabere ukupnu statistiku, poziva se metoda `prikaziUkupnuStatistiku` koja inicijalizira prazna polja sa: imenom festivala, prihodom festivala i još 2 pomoćna polja u koja se upisuju boje pozadine i boje obruba stupaca. Zatim se šalje AJAX zahtjev prema pomoćnoj skripti koja kao odgovor vraća sve festivale sa njihovim prihodima u JSON obliku. Kroz takav odgovor je moguće iterirati pomoću jednostavke `for` petlje i puniti svako polje sa potrebnim vrijednostima. Pomoćna polja sa bojama mogu se popuniti sa proizvoljnim bojama, a u ovom slučaju je odabrana crvena za pozadinu i tamnocrvena kao obrub. Graf se crta pomoću naredbe `new Chart` kojem se kao parametre proslijeđuju tip grafa i setovi podataka, odnosno polja koja su prethodno popunjena.

```
var canvasPrihod = $("#grafPrihod");
self.prikaziUkupnuStatistiku = function() {
        var p_oznake = [];
        var p_prihodi = [];
        var p_bojaPozadine = [];
        var p_bojaObruba = [];

        $.ajax({
        url: 'obrada-zahtjeva.php',
        method: 'POST',
        dataType: 'json',
        data: {
            key: 'dohvati-ukupnu-statistiku-prihod',
        }, success: function (response) {
                for(naziv in response){
                    p_oznake.push(naziv);
                    p_prihodi.push(response[naziv]);
                    p_bojaPozadine.push('rgba(75, 192, 192, 0.2)');
                    p_bojaObruba.push('rgba(75, 192, 192, 1');
                        }

                    if (grafPrihod) {
```

```
                grafPrihod.destroy();
            }

            var grafPrihod = new Chart(canvasPrihod, {
                type: 'bar',
                data: {
                    labels: p_oznake,
                    datasets: [{
                        label: 'Prihod (HRK)',
                        data: p_prihodi,
                        backgroundColor:
                         ↪ p_bojaPozadine,
                        borderColor: p_bojaObruba,
                        borderWidth: 1
                }]
            }
            });
        }
    })
```

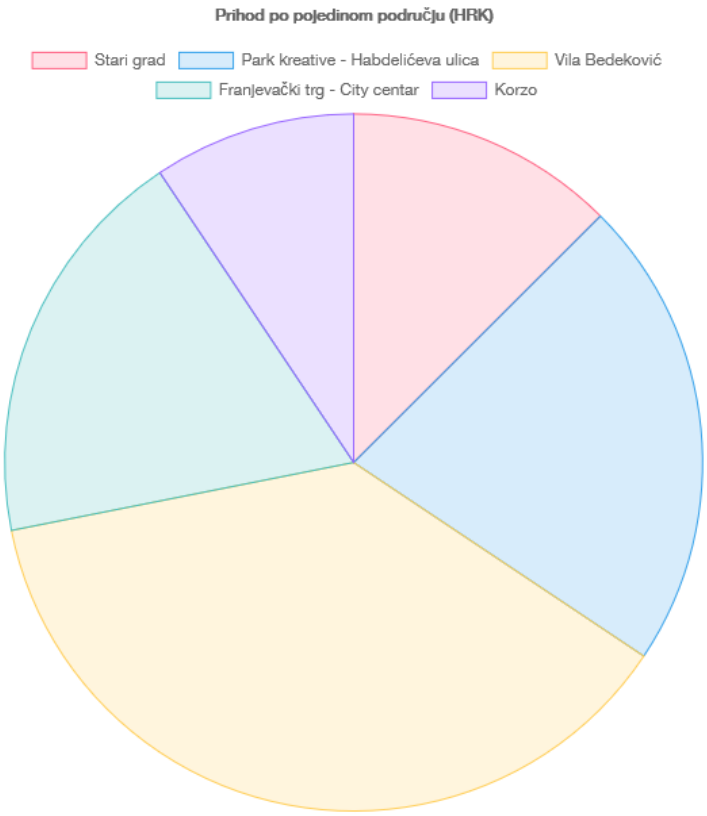
Valja svakako napomenuti da prije crtanja svakog grafa treba prvo provjeriti da li on već postoji, i ako postoji očistiti prikaz sa `destroy()` naredbnom. U suprotnom, novi graf bi se svaki put ispisivao preko starog i nastale bi pogreške u radu. Ispisani graf sa ukupnom statistikom festivala trebao bi izgledati ovako:



Slika 20: *Graf sa ukupnom statistikom festivala* (vlastita izrada)

Postupak za kreiranje grafa u obliku krofne/pite (engl. *pie*) je vrlo sličan, samo što je svaki kružni

isječak obojan drugom bojom, pa je prethodno kreirano statičko polje sa duginim pojama za pozadine i nešto tamnije nijanse za obrube. Graf sa prikazom prihoda po pojedinom području festivala bi trebao izgledati ovako:



Slika 21: *Graf sa pojedinačnom statiskikom festivala* (vlastita izrada)

8. Zaključak

U današnje vrijeme, web razvoj podrazumijeva poznavanje mnogo alata i tehnologija neovisno o tome radi li se o korisničkoj ili poslužiteljskoj strani. Svaki dan se pojavljuju nove tehnologije koje će "donijeti velike promjene" u web razvoju i promjeniti Web kakav ljudi sad poznaju. Zajedno sa tehnologijama, razvila su se i korisnička sučelja koja sve više zahtjevaju korisničku interakciju, ali ju znatno olakšavaju. Time se poboljšava korisničko iskustvo i zadovoljstvo, a to je zapravo jedino bitno kod današnjih web stranica i aplikacija. Upravo zbog takvih potreba svaki se dan pojavljuju novi JavaScript programski okviri i biblioteke za razvoj bogatih i interaktivnih korisničkih sučelja. Knockout.js je jedna od takvih biblioteka jer odvaja poslovnu logiku od modela podataka sa svojom MVVM arhitekturom, a sa konceptima promatranih varijabli korisničko sučelje čini interaktivnijim i bogatijim. To naročito dolazi do izražaja kod aplikacija sa web trgovinom gdje je potrebna dinamika. Nadalje, razvojni programeri se specijaliziraju u pojedinim programskim okvirima jer su vrlo kompleksni i puno više od samog JavaScripta, a ažuriraju se svakodnevno. Početni razvojni programeri (engl. *junior developers*) trebaju neprestano učiti kako bi bili u toku sa modernim web razvojem, a osim znanja na jednoj strani potrebno je i poznavati ključne koncepte i arhitekturu one druge strane. Zato danas ima vrlo malo čistih "Front-end" ili čistih "Back-end" razvojnih programera nego se najviše spominju oni sa znanjem potpunog stoga tehnologije. Naprimjer, razvojni programeri na korisničkoj strani ne trebaju učiti poslužiteljski skriptni jezik zbog pojave Node.js koji omogućava pisanje JavaScripta na poslužiteljskoj strani. Prije 10 godina nije se mogla zamisliti aplikacija bez poslužitelja, a danas su postoje bezposlužiteljska (engl. *serverless* programska sučelja koja koriste računala samo za obavljanje operacija, a ne konstantno. Uz to, jednostrane aplikacije zamijenit će progresivne web aplikacije (engl. *progressive web applications*), a korisnička sučelja će biti preplavljena animacijama zbog trenda pokretnog korisničkog sučelja (engl. *motion UI*). Zaključno, moderni web razvoj širi se eksponencijalno i nitko ne zna kako će izgledati u budućnosti, ali jedno je sigurno - biti će u potpunosti drukčiji nego danas.

Popis literature

[1] B. Getting. (2007). Basic Definitions: Web 1.0, Web 2.0, Web 3.0, adresa: https://www.practicalecommerce.com/Basic-Definitions-Web-1-0-Web-2-0-Web-3-0 (pogledano 13. 9. 2018).

[2] J. Munro. (2015). Knockout.js - Building Dynamic Client-Side Web Applications, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebanstopol.

[3] D. Kermek. (2018). Internet i Web, Povijest Interneta i Weba, Razvoj Interneta i Weba, prezentacija sa predavanja, FOI.

[4] P. A. Laplante. (2007). What Every Engineer Should Know about Software Engineering, CRC Press, Taylor Francis Group, 6000 Broken Sound Parkway NW, Suite 300.

[5] Wikipedia. (2018). User Interface, adresa: https://en.wikipedia.org/wiki/User_interface (pogledano 13. 9. 2018).

[6] P. M. Heathcote. (2004). 'A' Level Computing, Payne-Gallway Publishers Ltd, 26-28 Northgate Street, Ipswich.

[7] I. Sommerville, *User interface design process*, 2000. adresa: https://image.slidesharecdn.com/userinterfacedesignsommervillebangaloreuniversity-120819073259-phpapp02/95/user-interface-designsommerville-bangalore-university-20-728.jpg (pogledano 13. 9. 2018).

[8] T. A. Powell. (2002). The Complete Reference: Web Design, 2nd Edition, McGraw-HIll Companies, Inc.

[9] J. N. Robbins. (2006). Web Design in a Nutshell: A Desktop Quick Reference, 3rd Edition, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

[10] *A Guide To Responsive Web Design*, Verve Search Ltd - 81-83 Victoria Road - Surbiton - Surrey - KT6 4NS, 2017. adresa: https://www.vervesearch.com/best-practice-guides/responsive-web-design/ (pogledano 13. 9. 2018).

[11] D. Kermek. (2018). CSS stilske upute - Formatiranje primjenom kaskadnih stilskih listova, prezentacija sa predavanja, FOI.

[12] V. Mashevskiy. (2017). Front-End Web Development, adresa: http://eztuir.ztu.edu.ua/jspui/bitstream/123456789/6472/1/129.pdf (pogledano 13. 9. 2018).

[13] C. Wodehouse. (2016). Back-End Technology: The Role of the Back-End Web Developer, Upwork, adresa: https://www.upwork.com/hiring/development/back-end-web-developer/ (pogledano 13. 9. 2018).

[14] E. Brown. (2016). Learn JavaScript: Add sparkle and life to your web pages, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

[15] P. Ballard. (2015). Teach Yourself JavaScript in 24 Hours, 6h edition, SAMS 800 East 96th Street, Indianapolis, Indiana, 46240 USA.

[16] (2015). Introduction to the DOM, Mozilla Developer Network, MDN Web Docs, adresa: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction (pogledano 13. 9. 2018).

[17] H. Q. L. Thuan Thai. (2003). .NET Framework Essentials, 3rd Edition, O'Reilly Associates, Inc., 1005 Gravenstein Highway North, Sebastopol.

[18] C. Wodehouse. (2016). Understanding Software Frameworks - Web Application Frameworks - What is a Framework?, Upwork, adresa: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction (pogledano 13. 9. 2018).

[19] D. Riehle. (2000). Framework Design: A Role Modeling Approach, Ph.DThesis, No. 13509. Zürich, Switzerland, ETH Zürich, 2000, adresa: https://riehle.org/computer-science/research/dissertation/diss-a4.pdf (pogledano 13. 9. 2018).

[20] J. O'Neill, *A history of Javascript frameworks*, 2017. adresa: https://medium.com/@oneeezy/frameworks-vs-web-components-9a7bd89da9d4 (pogledano 13. 9. 2018).

[21] J. Smith. (2018). 9 Popular JavaScript Frameworks Used in 2018, Raygun, adresa: https://raygun.com/blog/popular-javascript-frameworks/ (pogledano 13. 9. 2018).

[22] E. M. Barnard. (2012). KnockoutJS Starter, Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK.

[23] J. Ferrando. (2015). KnockoutJS Essentials, Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK.

Popis slika

Popis tablica